Datenbank-Programmierung

Die Reihe Programmer's Choice

Von Profis für Profis
Folgende Titel sind bereits erschienen:

Bjarne Stroustrup
Die C++-Programmiersprache
1072 Seiten, ISBN 3-8273-1660-X

Elmar Warken
Kylix – Delphi für Linux
1018 Seiten, ISBN 3-8273-1686-3

Don Box, Aaron Skonnard, John Lam
Essential XML
320 Seiten, ISBN 3-8273-1769-X

Elmar Warken
Delphi 6
1334 Seiten, ISBN 3-8273-1773-8

Bruno Schienmann
Kontinuierliches Anforderungsmanagement
392 Seiten, ISBN 3-8273-1787-8

Damian Conway
Objektorientiertes Programmieren mit Perl
632 Seiten, ISBN 3-8273-1812-2

Ken Arnold, James Gosling, David Holmes
Die Programmiersprache Java
628 Seiten, ISBN 3-8273-1821-1

Kent Beck, Martin Fowler
Extreme Programming planen
152 Seiten, ISBN 3-8273-1832-7

Jens Hartwig
PostgreSQL – professionell und praxisnah
456 Seiten, ISBN 3-8273-1860-2

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
Entwurfsmuster
480 Seiten, ISBN 3-8273-1862-9

Heinz-Gerd Raymans
MySQL im Einsatz
618 Seiten, ISBN 3-8273-1887-4

Dusan Petkovic, Markus Brüderl
Java in Datenbanksystemen
424 Seiten, ISBN 3-8273-1889-0

Joshua Bloch
Effektiv Java programmieren
250 Seiten, ISBN 3-8273-1933-1

PEARSON Education

ADDISON-WESLEY

Andreas Herbolsheimer

Datenbank-Programmierung

Beispiellösungen mit Access, SQL Server und PostgreSQL

ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist bei
Der Deutschen Bibliothek erhältlich.

Umwelthinweis:
Dieses Produkt wurde auf chlorfrei gebleichtem Papier gedruckt.
Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem und recyclingfähigem PE-Material.

5 4 3 2 1

05 04 03 02

ISBN 3-8273-1945-5

ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10–12, D-81829 München/Germany

Einbandgestaltung: Christine Rechl, München
Titelbild: Malva neglecta, Malve. © Karl Blossfeldt Archiv – Ann und Jürgen Wilde, Zülpich/VG Bild – Kunst Bonn, 2002
Lektorat: Martin Asbach, masbach@pearson.de
CD-Mastering: Gregor Kopietz, gkopietz@pearson.de
Korrektorat: Christine Depta, München
Herstellung: Monika Weiher, mweiher@pearson.de
Satz: reemers publishing services gmbh, Krefeld, www.reemers.de
Druck und Verarbeitung: Bercker Graphischer Betrieb, Kevelaer
Printed in Germany

Inhalt

1 Einführung

Was mir an deinem System am besten gefällt? Es ist so unverständlich wie die Welt.
G. W. Hegel

Datenbanken sind mittlerweile in aller Munde. Geht es um Informationen für die Rasterfahndung, Messwerte für den Wetterbericht oder Angaben zu den Ausstellern einer Messe – es werden effiziente Datenbanken benötigt. Die Daten müssen in eine Datenbank eingegeben werden und bei speziellem Informationsbedarf sollen die benötigten Informationen sehr schnell verfügbar sein. Das 21. Jahrhundert schickt sich an, die Information als höchstes Gut auf Erden zu erheben. Weil dies so ist, drängt sich eine Frage auf. Wie kann die Masse an Informationen »gebändigt« werden? Jeder will auf dem Informationshighway der Schnellste sein. Der Ruf nach Informationssystemen, die den Menschen dabei unterstützen, ist unüberhörbar. Nun sind Systeme, wie sie die Datenbanken darstellen, keine Erfindungen der letzten Jahre, sondern sie gibt es bereits seit nahezu einem halben Jahrhundert. Trotzdem wachsen Verbreitung und Anforderungen bei der Datenbanktechnologie gleichermaßen rasant. Die anfänglich noch einfachen Modelle wurden erweitert und um neue Technologien und Konzepte ergänzt.

In diesem Buch sollen moderne Konzepte im Bereich der Datenbanktechnologie praxisnah vorgestellt werden. Mit einer ausgewogenen Mischung aus Theorie und Praxis richtet sich dieses Buch an die »Entscheider« und die »Macher« gleichermaßen. Programmbeispiele, Befehlsreferenzen und illustrative Modellübersichten helfen dem Programmierer bei der Entwicklung seines Systems und sie vermitteln dem Datenbankadministrator oder dem Projektmanager die nötigen Kenntnisse, die er zum Verständnis benötigt.

1.1 Der Einsatzbereich der Datenbank

Das Wort »Bank« ist uns im täglichen Sprachgebrauch als Stätte der wundersamen Geldvermehrung oder auch des Geldverlustes bekannt. So wie wir unser Geld zur Bank bringen und es dort für einen gewissen Zeitraum deponieren, um dann später einen Mehrwert zu erlangen, so in etwa verhält es sich auch mit der Datenbank. Üblicherweise wird in der Datenverarbeitung nicht mit flüchtigen Daten gearbeitet, son-

dern die Daten sollen permanent verfügbar sein. Daten, die einmal eingegeben wurden, sollen zu einem späteren Zeitpunkt abgerufen und bearbeitet werden. Diese Daten können in der Eingabereihenfolge auch abgespeichert werden.

Bei der amerikanischen Volkszählung im Jahr 1890 wurde dies in ähnlicher Form praktiziert. Die Volkszählung wurde mit einer elektromechanischen Zählmaschine des Ingenieurs *Herman Hollerith* bewältigt. Dieser so genannte *Hollerith Tabulator* fixierte die Daten auf Lochkarten in sequentieller Reihenfolge. Entsprechend groß waren die geometrischen Ausmaße und entsprechend lang dauerte auch die gesamte Volkszählung.

Das ist natürlich nicht mehr Stand der modernen Technik. Bereits jedes Schulkind kennt heute diese kleinen praktischen Taschendatenbanken, in denen sie einfach den Namen eingeben und unmittelbar darauf erscheint die dazugehörige Nummer. Und das ist noch nicht alles, sie können ihre Suche derart erweitern, dass sie nur den Anfangsbuchstaben und Ortsnamen angeben und trotzdem schnell zu ihrer gewünschten Telefonnummer, Faxnummer oder E-Mail-Adresse kommen. Diese Möglichkeiten sind für uns Standard geworden und rufen keine Begeisterungsstürme mehr hervor. Moderne Datenbanken müssen mehr leisten als Datenspeicherung und Datensuche. Es reicht nicht aus, Daten schnell zu finden, um sie dann modifizieren zu können.

Die Sekretärin eines mittelständischen Betriebes braucht die Adressdaten der Kunden nicht nur für Serienbriefe, die sie in einem Textverarbeitungsprogramm entworfen hat. Sie möchte auch wissen, welcher der Kunden sich in der letzten Zeit etwas rar gemacht hat und deshalb mal wieder ein Anschreiben braucht, um sich an das Unternehmen zu erinnern. Zusätzlich möchte sie wissen, in welchem Produktbereich der jeweilige Kunde bisher als Käufer aufgetreten ist und ob er eher konventionell seine Bestellung abgewickelt hat oder das E-Commerce-Angebot der Firma genutzt hat. Dementsprechend wird das Anschreiben auch auf herkömmlichem Weg über Post verschickt oder der Kunde erhält eine E-Mail, die das neuste Angebot enthält. Dieses Angebot wird selbstverständlich auch automatisch mit den aktuellen Produkt- oder Servicedaten aus der Datenbank zusammengestellt. Gleichzeitig werden die Internetseiten des Unternehmens über diese Datenbank aktualisiert.

Damit auch die anderen Niederlassungen des Unternehmens auf dem aktuellen Stand sind, wird die Datenbank regelmäßig repliziert, d.h. die Datenbanken der verschiedenen Standorte werden synchronisiert.

Anhand des angeführten Szenariums, lassen sich bereits einige Anforderungen an eine Datenbank erkennen. Leicht einzusehen ist nun sicherlich auch die Tatsache, dass ein Kleinunternehmer für seine Adressen nicht eine Datenbank benötigt, die mit den Möglichkeiten ausgestattet ist, wie sie von einem Wetterdienst verwendet wird.

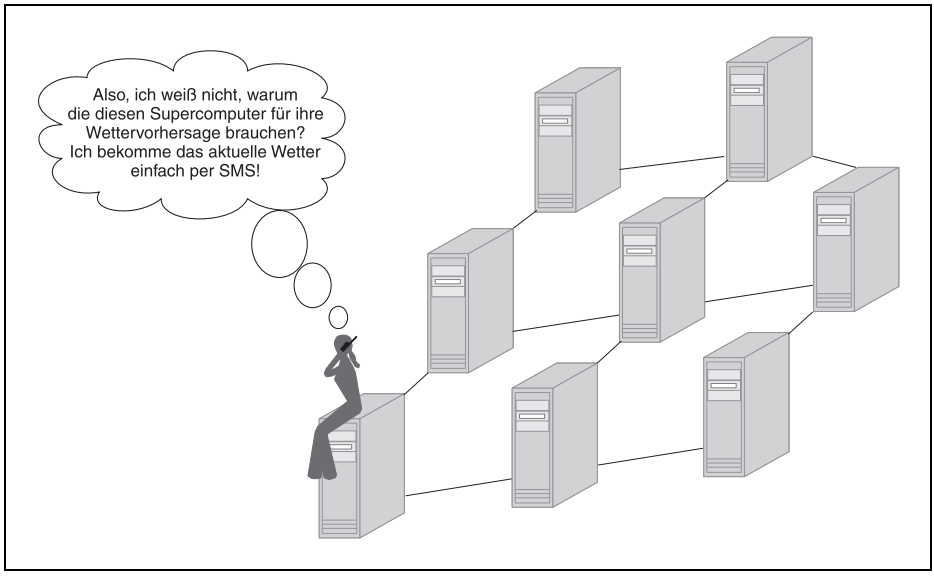


Abbildung 1.1: Unverständlich wie die Welt

Bei der Einsatzplanung der Datenbank muss auch irgendwann die Frage gestellt werden, ob der Aufwand für die Implementierung und Administration wirklich gerechtfertigt ist. Welche Datenbank erfüllt den angestrebten Zweck mit minimalem Aufwand?

1.2 Nutzbarkeit des Buches

Für den Leser ist von primärer Wichtigkeit der Informationsgewinn, den er aus einem Sachbuch ziehen kann. Da jeder Leser mit einem anderen Basiswissen ausgestattet ist, kann nicht von einem Standard an Vorkenntnissen ausgegangen werden. Aus diesem Grund wird dem Leser all das vermittelt, was er für den praktischen Umgang mit Datenbanken benötigt. Elementare Grundlagen müssen für das Verständnis von praktischen Umsetzungen behandelt werden und werden deshalb in kleinen »verdaulichen Happen« serviert.

Die praktische Komponente, d.h. die Beispieldatenbank, zieht sich wie ein roter Faden durch die einzelnen Kapitel des Buches. Die verschiedenen Tätigkeiten, die notwendig sind, um eine Datenbankanwendung zu erstellen, finden sich chronologisch in den entsprechenden Kapiteln wieder. So kann der Leser sehr schnell, auch durch Überlesen bereits bekannter Thematiken, zu seiner Problemlösung kommen.

Damit wirklich ein breites Anwendungsfeld abgedeckt wird, sind weit verbreitete Technologien und Standards Hauptbestandteil der einzelnen Kapitel. Es wird jedoch

an geeigneter Stelle auch immer wieder ein Blick in den teilweise unübersichtlichen Datenbankdschungel gewagt. Das Verwenden des Opensource-DBMS PostgreSQL soll auch das Bestreben nach »Lösungssuche jenseits der kommerziellen Pfade«, widerspiegeln und die ansonsten etwas »windows-lastigen« Lösungen ergänzen. Mit der Auswahl verschiedener DBMS-Anbieter wird zusätzlich das zugegebenermaßen waghalsige Ziel verfolgt, allgemeingültige Lösungen vorzustellen. Das Anpassen an eigene Anforderungen sollte daher kein allzu großes Problem sein.

Grundlegende Tätigkeiten, wie das Erstellen der Datenbank und ihrer einzelnen Komponenten, sind für die drei vorgestellten Datenbanken *MS Access*, *MS SQL Server* und *PostgreSQL* beschrieben. Bei weiterführenden Techniken, wie *Data Warehousing* oder *Internetanbindung*, wurde die Lösung gewählt, mit der Sie am schnellsten zum Ziel kommen.

Das Buch widmet sich in den einzelnen Kapiteln vorrangig der *Funktionalität* von Datenbanksystemen. *Gestaltungsrichtlinien* für Benutzeroberflächen oder Berichte, die als kaufentscheidendes Kriterium oftmals unterschätzt werden, finden aber auch Berücksichtigung in den jeweiligen Kapiteln.

Für denjenigen, der sich völlig neu in die Datenbanktheorie einarbeitet, werden viele Begriffe neu sein. *Englische Fachbegriffe* wurden in Klammern mit einem deutschen Begriff erklärt. Ebenso wurden eingedeutschte Bezeichnungen mit den englischen Übersetzungen versehen. Üblicherweise werden die Fachbegriffe direkt bei ihrer Verwendung kurz erklärt. Gibt es darüber hinaus Begriffe, deren Bedeutung Ihnen nicht geläufig ist, steht Ihnen im Anhang ein umfangreiches Glossar zur Verfügung.

Mit einer Vielzahl von Beispielen, die sich überwiegend auf die *Beispieldatenbank* beziehen, werden die theoretischen Erklärungen verständlich gemacht.

Die folgende Stichwortliste gibt eine Zusammenfassung der vermittelten Konzepte und Aufgaben in den einzelnen Kapiteln dieses Buches:

- Auswahl der Komponenten und Kaufentscheidung (Kapitel 2),
- Systemplanung (Kapitel 2 und 3),
- Datenbankentwurf (Kapitel 3),
- Arbeiten mit der Abfragesprache SQL (Kapitel 4),
- Aufbauen der Datenbank und ihrer Objekte (Kapitel 5),
- Erstellen von Eingabemasken (Kapitel 6),
- Verbindung zwischen Datenbank und Anwendung herstellen (Kapitel 7),
- Datenbank verwalten und optimieren (Kapitel 8),
- Einrichten von Sicherheitsmechanismen (Kapitel 9),

- Wartung der Datenbank (Kapitel 9),
- Erstellung eines Datenbankberichtes (Kapitel 10),
- Unternehmensdaten zusammenfassen und analysieren (Kapitel 11),
- Internetanbindung der Datenbank (Kapitel 11),
- Arbeiten mit Multimedialen Daten (Kapitel 12).

1.3 Von der Theorie zur Praxis: Die Beispieldatenbank

Damit der Titel des Buches nicht nur eine Phrase bleibt, soll schon in der Einführung der erste Schritt zum praktischen Einstieg gemacht werden. Anhand einer Beispieldatenbank aus dem Dienstleistungssektor werden in diesem Buch sämtliche theoretischen Grundkenntnisse und programmspezifischen Kniffe behandelt. Um den heutigen Anforderungen eines Unternehmens an eine Datenbank zu entsprechen, werden moderne Konzepte, unabhängig vom gültigen SQL-Standard, beschrieben. Ob beim Arbeiten mit verteilten Daten (auf mehrere Standorte) oder beim Veröffentlichen der Daten im Internet, wird die Beispieldatenbank die Basis für Problemlösungen sein. Dies hat den Vorteil, dass der Leser sich nicht wieder in neue Szenarien hineindenken muss und daher schließlich eine Datenbank hat, deren Zustandekommen er Schritt für Schritt erarbeiten konnte. Eine Modifizierung der Datenbank oder ein Abbilden auf andere Bereiche sollte dann problemlos möglich sein. Selbst das Aufbauen einer Datenbank auf anderen Plattformen wird nach dem Studieren des Buches nicht mehr länger eine unüberwindbare Klippe sein.

Unsere Beispieldatenbank soll für ein Unternehmen eingerichtet werden, das EDV-Dienstleistungen anbietet. Dieses Unternehmen hat Niederlassungen in Hamburg, Köln und Berlin. Aus den Servicedaten der Datenbank wird automatisiert ein Katalog erstellt und in die firmeneigenen Internetseiten gespeist. Die Zentrale des Unternehmens befindet sich in Berlin. Hier werden auch die Servicedaten über Eingabemasken erfasst. Die anderen Niederlassungen sollen nicht die Möglichkeit haben, neue Servicedaten in der Datenbank aufzunehmen. Der Kundenstamm kann sowohl in der Zentrale, als auch in den Niederlassungen erweitert werden. Außerdem werden Kunden, die ihre Bestellung online aufgeben, ebenfalls in dieser Datenbank erfasst. Die Marketingabteilung braucht für die Analyse des Kaufverhaltens und zur Entscheidungsfindung spezielle OLAP-Funktionen, die im Kapitel 11 beschrieben werden.

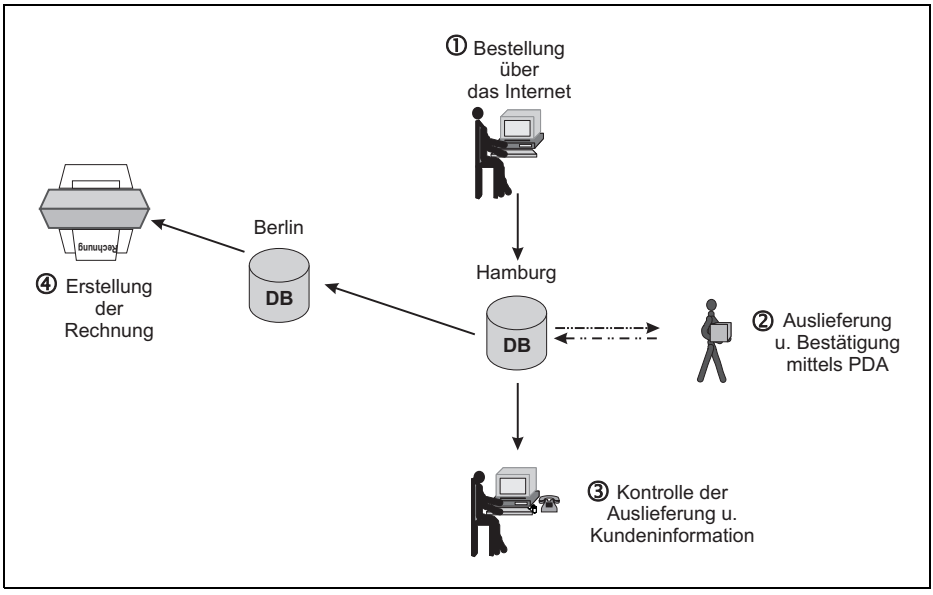


Abbildung 1.2: Szenarium für die Beispieldatenbank

Falls Sie sich entscheiden, mit PostgreSQL zu arbeiten, so können Sie die Installationsdateien der Begleit-CD verwenden und die Installation, wie in Anhang F beschrieben, ausführen.
Möchten Sie hingegen mit dem SQL Server arbeiten, können Sie sich unter folgender Webadresse *www.microsoft.com/germany/sql/bucheval* anmelden und erhalten kostenlos eine 120-Tage-Testversion der MS SQL Server 2000 zugesandt.

2 Allgemeine Konzepte

Den Praktiker interessieren üblicherweise theoretische Konzepte nur dann, wenn sie ihm unmittelbar bei seinem konkreten Anwendungsfall weiterhelfen. Ganz ohne theoretische Kenntnisse geht es selbstverständlich nicht. In diesem Kapitel wird deshalb all das zusammengetragen, was der Programmierer, Administrator oder IT-Entscheider benötigt, um sein Softwareprojekt in die richtigen Bahnen zu leiten. Bevor theoretische Konzepte praktisch umgesetzt werden können, muss eine gewisse Grundkenntnis der Datenbanktheorie vorhanden sein. Dieses Kapitel bietet neben den Grundkenntnissen einen Überblick über verschiedene Datenbankplattformen und Programmiersprachen, mit denen diese angesteuert werden können.

Der Datenbankprogrammierer ist bei der steigenden Komplexität und internationaler Verbreitung der Applikationen angehalten, sich über international gültige Normierung Gedanken zu machen und die Regeln des Qualitätsmanagements zu beachten. Auch zu diesen Stichworten gibt es eine Einführung innerhalb dieses Kapitels.

2.1 Datenmodelle

Das theoretische Konzept für die Strukturen zur Datenhaltung in einer Datenbank bilden die Datenmodelle. Die vier bedeutsamsten Modelle in der chronologischen Reihenfolge sind:

- Hierarchisches Modell,
- Netzwerkmodell,
- Relationales Modell,
- Objektorientiertes Modell.

Am Beispiel der Mitarbeiterdaten eines Unternehmens sollen die verschiedenen Modelle illustriert werden.

2.1.1 Hierarchisches Modell

Der hierarchische Ansatz ist das älteste Konzept für Datenmodelle und ist eng mit der Entwicklung des Datenbanksystems IMS der Firma IBM verbunden. Die Struktur in einer hierarchischen Gliederung sieht folgendermaßen aus:

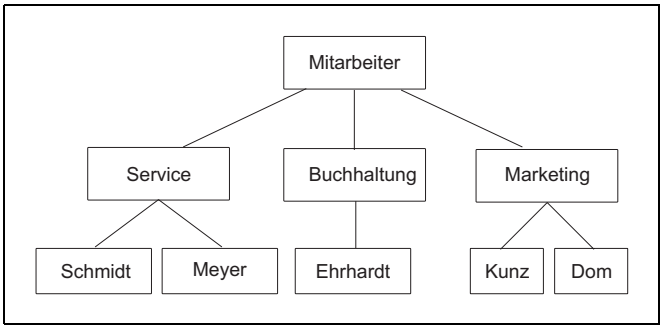


Abbildung 2.1: Beispiel der Mitarbeiterdaten im hierarchischen Modell

Es ist leicht erkennbar, dass die Daten im Sinne eines Baumgraphen angeordnet sind. In drei Ebenen (Hierarchien) sind die Mitarbeiterinformationen eines Unternehmens dargestellt. Dies könnte die Teilstruktur einer Adressenverwaltung sein. Auf der ersten Ebene hätten wir die Information über den Adressentypus, in diesem Fall Mitarbeiter. Die zweite Ebene liefert die Information über die Abteilung des Mitarbeiters. Schließlich erhalten wir in der dritten Ebene den Namen des Mitarbeiters. Zwei typische Merkmale lassen sich anhand des Baumgraphes ableiten:

1. Die Elemente der verschiedenen Ebenen stehen zueinander in *1:1*- oder *1:n*-Beziehungen, d.h. ein Element einer höheren Ebene kann zu einem oder beliebig vielen Elementen einer unteren Ebene stehen.
2. Die Datensätze sind immer vom Wurzeltyp. Es kann keinen Namen ohne Angabe der Abteilung bzw. keine Abteilung ohne Angabe des Mitarbeitertypus geben.

Diese Einschränkungen lassen das Umsetzen realer Zusammenhänge in Form des hierarchischen Modells nicht immer zu.

2.1.2 Netzwerkmodell

Das Netzwerkmodell kann als Weiterentwicklung des hierarchischen Modells angesehen werden. Im Netzwerkmodell ist es möglich, *m:m*-Beziehungen darzustellen. Netzwerke können auch als Modelle mit mehreren Hierarchien betrachtet werden, d.h. ein *Sohnelement* (Element einer unteren Ebene) kann mehr als ein *Vaterelement* (Element einer übergeordneten Ebene) haben. Die Struktur eines Netzwerkmodells könnte ein Aussehen wie in Abbildung 2.2 haben.

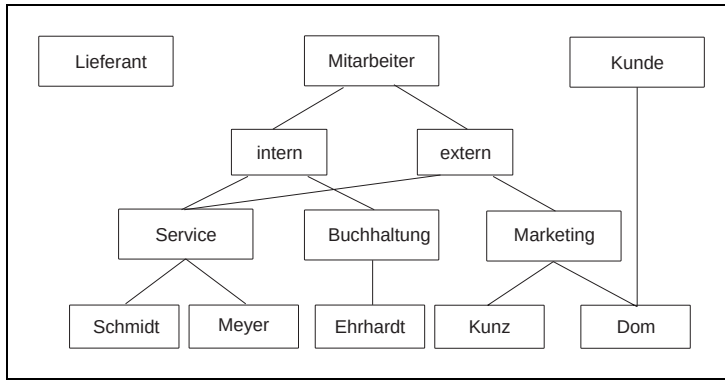


Abbildung 2.2: Beispiel der Mitarbeiterdaten im Netzwerkmodell

Zwischen der Typus- und Abteilungsebene wurde noch eine zusätzliche Ebene eingeführt, welche die Mitarbeiter in externe und interne aufteilt. Die Abteilung Service ist in diesem Beispiel nicht mehr nur einem, sondern zwei Vaterelementen zugeordnet. Außerdem ist es möglich, dass ein Mitarbeiter auch gleichzeitig Kunde ist. Wir haben bei diesem Modell also mehr Flexibilität, was die Umsetzung der Strukturen aus der Realwelt anbelangt. Als Beispiele für kommerziell vertriebene netzwerkartige Datenbanksysteme sind UDS von Siemens oder DMS von DEC zu nennen.

2.1.3 Relationales Modell

Das Modell nach *E.F. Codd* benutzt die *Tabelle* als Datenstruktur. Es werden nur Daten und keine Beziehungen zwischen diesen dargestellt. Durch einen Vergleich von verschiedenen Daten kann aber eine Beziehung rekonstruiert werden.

Tabelle: Abteilung

AbtNr	Name
1	Service
2	Buchhaltung
3	Marketing

Tabelle: Mitarbeiter

MitarbNr	AbtNr	Int_Ext
1001	1	I
1002	1	E
1003	2	I
1004	3	E
1005	3	E

Tabelle: Adresse

AdrNr	MitarbNr	Name	Ort
221	1001	Schmidt	Hamburg
222	1002	Meyer	Berlin
223	1003	Ehrhardt	Berlin
224	1004	Kunz	Berlin
225	1005	Dom	Bremen

Abbildung 2.3: Beispiel der Mitarbeiterdaten im relationalen Modell

Diese Tabellen haben eine zweidimensionale Struktur, d.h. die Informationen werden in Datensätzen, also in Sammlungen von Einzelinformationen zeilenweise abgelegt. In einer Spalte sind Informationen der gleichen Art.

In unserem Beispiel sind die Daten zunächst nur in den Tabellen *Abteilung*, *Mitarbeiter* und *Adresse* abgelegt. Die Beziehungen der Daten zueinander, können über so genannte Schlüsselfelder (hier die Felder *AbtNr* oder *MitarbNr)* aufgebaut werden.

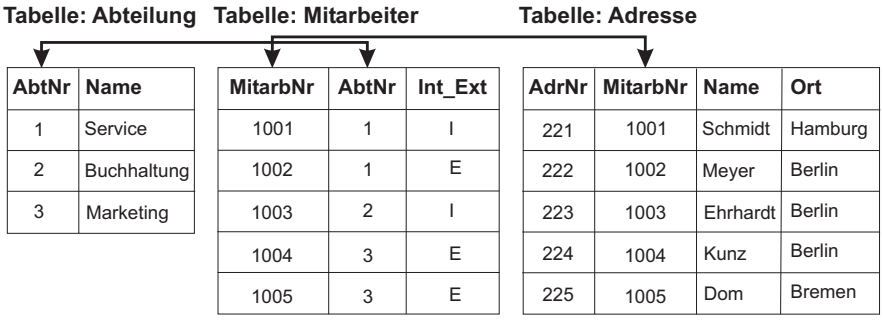


Tabelle: Abteilung

AbtNr	Name
1	Service
2	Buchhaltung
3	Marketing

Tabelle: Mitarbeiter

MitarbNr	AbtNr	Int_Ext
1001	1	I
1002	1	E
1003	2	I
1004	3	E
1005	3	E

Tabelle: Adresse

AdrNr	MitarbNr	Name	Ort
221	1001	Schmidt	Hamburg
222	1002	Meyer	Berlin
223	1003	Ehrhardt	Berlin
224	1004	Kunz	Berlin
225	1005	Dom	Bremen

Abbildung 2.4: Schlüsselfelder im relationalen Modell

Schlüsselfelder, vergleichbar mit einer Hausnummer, werden in einer Tabelle benutzt, um jeden Datensatz innerhalb der Tabelle mit einem eindeutigen Identifikationsmerkmal zu versehen. In der Tabelle *Mitarbeiter* ist das Feld *MitarbNr* ein solches Schlüsselfeld. Diese Spalte hat für jeden Datensatz einen eindeutigen Wert. Mehrfachnennungen sind nicht möglich. Auf diese Weise kann dem Mitarbeiter einfach eine Abteilung oder eine (oder mehrere) Adresse zugewiesen werden. Weitere Eigenschaften könnte man den bereits vorhandenen Tabellen hinzufügen.

Sie sehen, das relationale Modell hat eine einfache, schnell überschaubare Struktur. Die einfache Struktur des relationalen Modells kann jedoch auch von Nachteil sein. Soll ein Anwendungsobjekt als Ganzes bearbeitet werden, das über mehrere Tabellen verteilt ist, muss jeweils eine Verknüpfung erzeugt werden. Bei komplexen Objekten kann dies sehr aufwändig sein. Bei der Aufteilung in mehrere Tabellen werden künstliche Schlüsselfelder angelegt. Diese entsprechen nicht den realen Eigenschaften und bedeuten deshalb einen Mehraufwand an Verwaltungsarbeit.

Soll es darum gehen, im dreidimensionalen Raum Bewegungen zu beschreiben, stößt man bei diesem Modell schlichtweg auf Grenzen.

Das relationale Datenmodell ist jedoch dank seiner eingängigen, spartanischen Datenstruktur bei gleichzeitiger Existenz von mächtigen, mengenorientierten Operatoren ein weit verbreitetes Datenmodell (besonders im betrieblichen Bereich).

Mathematische Wurzeln des relationalen Modells

Die mathematischen Wurzel des relationalen Datenmodells finden wir in der Relationenalgebra. Eine Relation wird hier wie folgt definiert:

Ein Objekt kann mehrere Eigenschaften haben, die wird im Folgenden *Attribute* nennen wollen. Es sei A = {a1, a2,...,an} eine endliche Menge von Attributen, wobei jedes Attribut einen Wertebereich besitzt. Man nennt diesen auch *Domäne*, dom(a). Bei dem Wertebereich handelt es sich um atomare Werte z.B. Integer, String, Boolean etc.

Eine n-stellige Relation , R über A wäre dann eine Teilmenge des kartesischen Produktes über den Attribut-Domains der zugrundeliegenden Attributmenge A:

$R \subseteq dom(a1)\ x\ ...\ x\ dom(an)$

Ein Element einer Relation heißt Tupel zu einer Relation R über A und entspricht genau einer Zeile in der Tabelle, welche diese Relation darstellt.

Schließlich heißt dies praktisch nichts anderes, als dass die Eigenschaften eines Objektes, wie es der Mitarbeiter in unserem Beispiel war, das Objekt beschreibend, in einer (oder mehreren) Tabelle zusammengefasst werden können.

Die Verknüpfung von zwei Tabellen, die wir bereits oben in der Anwendung kennen gelernt haben, wird in der Relationenalgebra *natürlicher Verbund* genannt.

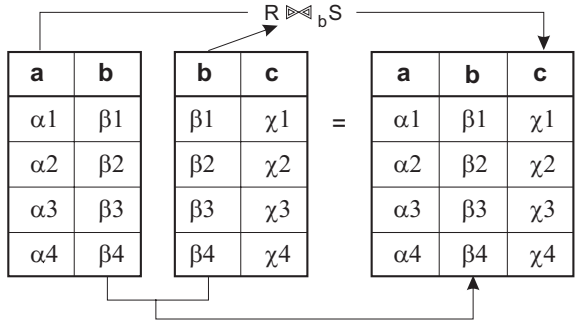


Abbildung 2.5: Natürlicher Verbund – Beispiel

Das Ergebnis eines natürlichen Verbundes (engl. Natural Join) der Relation R, S über Attribute a von R und b von S enthält das kartesische Produkt aller Tupel von R, S, die in a bzw. b die gleichen Werte enthalten, wobei die »doppelte Spalte« nur einmal auftaucht.

Es seien R(a1, a2,...,am,a) ∈ Rel(A), S(b,b1,b2,...,bn) ∈ Rel(B), dom(a) = dom(b), so ist

$R \bowtie a{=}b\ S := \{ (\alpha 1, \alpha 2,.., \alpha m, \beta, \beta 1, \beta 2,..., \beta n) \mid (\alpha 1, \alpha 2,.., \alpha m, \beta) \in R,$
$(\beta, \beta 1, \beta 2,..., \beta n) \in R\}$

Sind a und b namensgleich, so schreiben wir auch kurz: $R \bowtie b\ S$

Weitere gängige Operatoren der Relationenalgebra sind in der folgenden Tabelle zusammengefasst.

Operator	**Abkürzung/Symbol**
Datengruppierung	Grp
natürlicher Verbund zweier Relationen	Join
Projektion	Proj
Umbenennung von Attributbezeichnungen	Ren
Auswahl	Sel
Kreuzprodukt zweier Relationen	X
Vereinigungsmenge	$\cup$
Durchschnittsmenge	$\cap$
Differenzmenge	\
Vergleichsoperatoren	$=,\neq,\geq,<,...$
logische Operatoren	$\wedge,\vee,\neg...$

Tabelle 2.1: Operatoren aus der Relationenalgebra

In einer weiteren Tabelle soll nochmals ein kurzer Überblick über die einzelnen Bezeichnungen und ihre Herkunft erfolgen:

Relationenalgebra	**SQL**	**Englische Bezeichnung**
Relation	Tabelle	table
Tupel	Zeile/Datensatz	row/record
Attribut	Spalte	column
Domäne	Wertebereich	values

Tabelle 2.2: Definitionsüberblick für das relationale Datenmodell

2.1.4 Objektorientierte Modelle

Auch das relationale Modell hat, wie bereits angedeutet, seine Schwachpunkte. Ein weiterer evolutionärer Schritt bei der Modernisierung der Datenbanktechnologie ist das Berücksichtigen des objektorientierten Modells. Die Idee der objektorientierten Datenverarbeitung verbreitet sich seit Beginn der 80er Jahre und gewinnt sowohl in Programmiersprachen als auch im Datenbankbereich weiterhin an Bedeutung.

Das objektorientierte Datenmodell kann um anwendungsspezifische Datentypen und Funktionen erweitert werden. Und schließlich gibt es Datendefinitions- und Datenmanipulationssprachen, die für das objektorientierte Modell zugeschnitten sind.

Das objektorientierte Datenmodell bietet:

1. Die üblichen Standarddatentypen, die vom System vordefiniert sind. Das sind zum Beispiel die wohlbekannten Integer-, String-, oder Character-Datentypen.
2. Die typischen Konzepte der Objektorientierung: Klassen und Methoden, d.h. Operationen auf den Objekten der Klasse.
3. Konzepte zur Beschreibung komplexer Objekte. Dazu werden Typkonstruktoren in die Welt der Datenbanken eingeführt. Es handelt sich dabei um Sprachelemente, mit denen komplexe Datentypen aufgebaut werden können.

In einem semantischen Datenmodell (SDM) soll an dieser Stelle für unser Standardbeispiel ein objektorientiertes Modell erstellt werden.

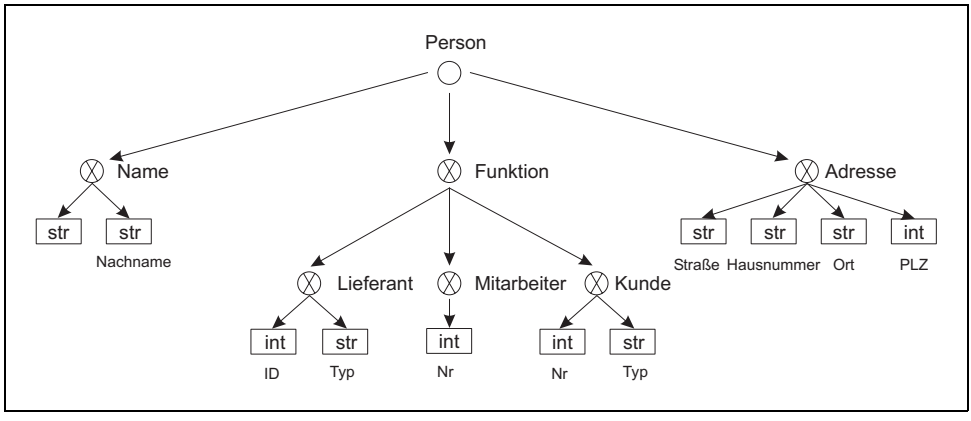


Abbildung 2.6: Beispiel der Mitarbeiterdaten im objektorientierten Modell (SDM)

Der Objekttyp *Person* wird durch Aggregationen (⊗) und Standarddatentypen (□) dargestellt. Bei der Aggregation wird ein neuer Untertyp erzeugt, der die eingehenden Obertypen als Komponenten beinhaltet. Der Untertyp *Mitarbeiter* hat also als Komponenten *Funktion* und *Person*. Beim objektorientierten Ansatz nennt man diesen Zusammenhang auch Vererbung der Eigenschaften. Die Eigenschaften eines Typs werden zusammen mit dem zugehörigen Datentyp vererbt.

Ein wichtiges Merkmal objektorientierter Modelle gegenüber dem klassischen relationalen Datenmodell ist die Objektidentität. Jedes Objekt in der Datenbank erhält bei seiner Erstellung vom Datenbanksystem einen Identifikator, so etwas wie einen Eindeutigkeitsstempel. Der Objektidentifikator ist systemweit eindeutig und ändert sich während der Lebenszeit des Objektes nicht.

Ein objektorientiertes Typsystem für Programmiersprachen besteht aus Klassen, Objekten, Methoden, Spezialisierungsbeziehungen. Das wird über objektorientierte

Datenbanken in die Datenbankwelt übertragen; allerdings sind die Wege dahin unterschiedlich. Standards sind für jede neue Technologie sehr wichtig.

Im Bereich objektorientierter Datenbanken sind zwei Standardisierungsrichtungen maßgebend. Im Rahmen von der Datenbanksprache SQL-99 wurden neue objektorientierte Sprachkonstrukte eingeführt. Die kommerziellen Datenbankanbieter versuchen seit Anfang der 90er Jahre, die unterschiedlichen objektorientierten Datenmodelle und Sprachen ihrer Systeme zu vereinheitlichen. Dazu haben sie sich zu einer Standardisierungskommission zusammengeschlossen, der ODMG (Object Data Management Group). Die von ODMG definierten Standards betreffen ein einheitliches Objektmodell, objektorientiertes Metamodell, sprachkonsistente Implementationsvorschriften und eine Objektdefinitionssprache. OQL (Object Query Language) ist eine objektorientierte Abfragesprache, die eigenständig genutzt oder eingebettet in Wirtssprachen verwendet werden kann. ODL (Object Definition Language) ist eine Definitionssprache für das Objektmodell. Die Verarbeitung der Daten geschieht durch die sprachabhängige OML (Object Manipulation Language).

Auf dem Markt der kommerziellen Datenbanksysteme sind nun sowohl objektorientierte als auch objektrelationale Produkte zu finden. Bei den objektrelationalen Produkten handelt es sich um erweiterte relationale Datenmodelle. Objektorientierte Produkte basieren vollständig auf dem objektorientierten Modell.

2.2 Relationale Datenbank-Managementsysteme

Das relationale Datenbank-Managementsystem, welches wir im weiteren Verlauf abkürzend *RDBMS* nennen wollen, ist eine Ansammlung von Werkzeugen, welche die Daten nach dem relationalen Modell verwalten.

Hierzu gehört im einfachsten Anwendungsfall das Lesen und Schreiben der Daten. In Abbildung 2.7 wird das Prinzip einer Datenbankanwendung gezeigt. Der Benutzer, der an seinem PC sitzt (auch Client genannt) fordert aus der Datenbank spezielle Informationen. Bei diesem Vorgang wird vom Client eine oder mehrere Abfragen (SQL-Anweisungen) an den Server geschickt. Im Normalfall geschieht dies jedoch im Verborgenen. Der Benutzer klickt in seinem Programm auf Befehlsschaltflächen und Menüs, hinter denen sich ein Programmiercode verbirgt. Während der Ausführung des Codes wird eine Abfrage an den Datenbankserver übermittelt. Vom Datenbankserver, auf dem die Daten physikalisch und logisch abgelegt sind, wird dann eine Ergebnismenge an den Client gesendet. Dort wird das Ergebnis nach Bedarf umgeformt. Dies kann z.B. in Listenform zur Ausgabe über einen Netzwerkdrucker geschehen. Moderne RDBMS bieten jedoch darüber hinaus eine Vielzahl weiterer Funktionen, die man ohne weiteren Programmieraufwand abrufen kann. Dazu gehören Mittel für den Datenbankentwurf (siehe Kapitel 3), Module für die Erstellung der DB und ihrer

Objekte (siehe Kapitel 6), administrative Werkzeuge (siehe Kapitel 8 und 9) sowie Programmeinheiten für die Datenanalyse und das Publizieren von Daten im Internet (siehe Kapitel 11).

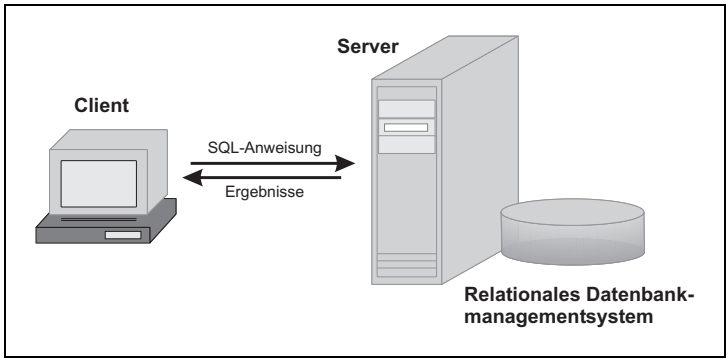


Abbildung 2.7: Prinzip einer Datenbankanbindung mit RDBMS als Plattform

Zunächst sollen aber einige elementare Komponenten eines RDBMS vorgestellt werden. Eine Datenbank ist auch eine Sammlung von Daten, Tabellen und sonstigen Objekten. Datenbankobjekte erleichtern sowohl die Strukturierung als auch die Definition von Mechanismen für das korrekte Lesen und Schreiben der Daten.

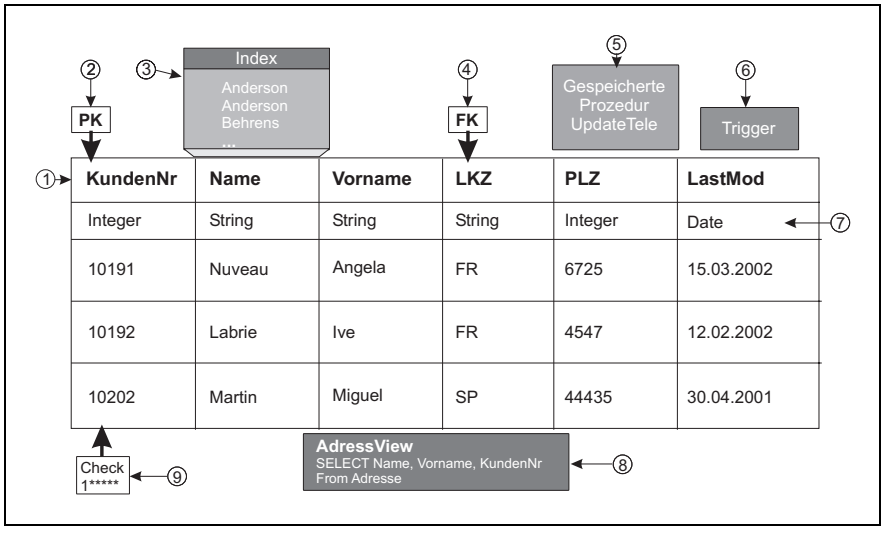


Abbildung 2.8: Objekte des RDBMS

Objekte innerhalb der Datenbank sind als Komponenten zu verstehen, die dem Anwender die Datenhaltung erleichtern und intern SQL-Definitionen umsetzen. In Abbildung 2.8 sind einige Objekte für die Tabelle *Adresse* dargestellt. Je nach DBMS werden diese Objekte mit SQL-Anweisungen oder aber über grafische Konfigurationsmöglichkeiten generiert. Im Folgenden nun eine Kurzbeschreibung der einzelnen Objekte.

1. Tabelle

Bei der Tabelle handelt es sich um eine zweidimensionale Struktur, in der Daten gespeichert werden. Eine Tabelle besteht aus Spalten und Zeilen.

2. u. 4. Primärer Schlüssel und Fremdschlüssel

Eine oder mehrere Spalten einer Tabelle können als Schlüssel definiert werden. Über die Schlüssel einer Tabelle kann eine Beziehung zu einer anderen aufgebaut werden. Der primäre Schlüssel der einen Tabelle steht hierbei in Beziehung zu dem Fremdschlüssel einer anderen. Die Wertebereiche der Fremd- und primären Schlüssel müssen gleich sein. In der abgebildeten Tabelle ist die Spalte *KundenNr* der primäre Schlüssel. Die Werte in dieser Spalte dienen auch der eindeutigen Identifikation des jeweiligen Datensatzes, daher dürfen keine Mehrfachnennungen vorkommen. Die Spalte mit der Spaltenüberschrift *LKZ* dient in dieser Tabelle als Fremdschlüssel. Über diesen Fremdschlüssel ist es möglich, eine Beziehung zu einer weiteren Tabelle herzustellen, die uns die Information über das Herkunftsland gibt, so wie wir das in Abbildung 2.9 sehen.

Tabelle: Kunde

KundenNr	Name	Vorname	LKZ
10191	Nuveau	Angela	FR
10192	Labrie	Ive	FR
10202	Martin	Miguel	SP

FK

PK

Tabelle: Land

LKZ	Bezeichnung
D	Deutschland
FR	Frankreich
SP	Spanien
NL	Niederlande

Abbildung 2.9: Verknüpfung von zwei Tabellen über Schlüsselfelder

In der zusätzlichen Tabelle *Land* haben wir auch wieder das Schlüsselfeld *LKZ*. Im Gegensatz zum gleichen Feld der Tabelle *Kunde* handelt es sich hierbei jedoch um den primären Schlüssel. Die Werte in dieser Spalte müssen eindeutig sein, damit es auch zu einer eindeutigen Zuordnung der Datensätze der beiden Tabellen kommen kann. Fremdschlüssel dürfen in einer Tabelle mehrfach verwendet werden, primäre Schlüssel jedoch nur einmal.

3. Index

Eine Speicherstruktur, die einen schnellen Zugriff für den Datenabruf ermöglicht und die Korrektheit (Integrität) der Daten in den Tabellen erzwingen kann.

5. Gespeicherte Prozeduren

Eine benannte Auflistung von SQL-Anweisungen, die zusammen ausgeführt werden. Die gespeicherte Prozedur entspricht einer Funktion, die bereits aus der Welt der Programmiersprachen bekannt sein dürfte.

6. Trigger

Eine spezielle gespeicherte Prozedur, die automatisch ausgeführt wird, wenn ein Benutzer Daten in einer Tabelle bearbeitet.

7. Datentyp

Definiert die für eine Spalte oder Variable zulässigen Datenwerte. In der Tabelle ist für die Spalte *Vorname* der Datentyp *String* vergeben, d.h. in dieser Spalte sind nur Werte zulässig, die diesem Typen entsprechen.

8. Sicht

Ermöglicht das Anzeigen von Daten aus einer oder mehreren Tabellen oder Sichten einer Datenbank. Da Informationen zu einem Objekt oder Sachverhalt sich oftmals über mehrere Tabellen erstrecken, ist dies eine Möglichkeit für eine Informationszusammenfassung.

9. Einschränkung

Es werden Regeln für die in Spalten zulässigen Werte definiert und die Korrektheit der Daten erzwungen. Im angegeben Beispiel wird überprüft, ob die erste Ziffer der *KundenNr* eine eins ist.

2.3 Welche Datenbank wofür – Marktanalyse

Bei den Vorbereitungen zu einem Softwareprojekt mit Datenbankanbindung stellt sich zwangsläufig die Frage, welche Datenbank für das Projekt einzusetzen ist. Wie so oft im Leben gibt es auch in diesem Fall kein Patentrezept. Es gibt einige Schlüsselkriterien, von denen man eine Kaufentscheidung abhängig machen sollte. Doch oftmals sind es subjektive Gründe, die eher in der Fimenpolitik oder bei den Vorlieben der Programmierer zu suchen sind, als dass zuvor eine objektive Marktanalyse stattgefunden hätte. Der Kunde, dem das System schließlich verkauft wird, ist sowieso überfordert die Vor- und Nachteile der Datenbank, die ihm mit seiner Anwendung untergejubelt wurde, vollständig zu erschließen. Der Datenbankprogrammierer sollte schließlich in

den Entscheidungsprozess eingebunden sein und deshalb auch über eventuelle Alternativen informiert sein. Der Markt bietet eine breite Palette an Produkten, die ganz verschiedene Zielgruppen ansprechen sollen. Deshalb werden bei der sorgfältigen Analyse einige Systeme aus dem Raster herausfallen, weil sie für den Anwendungsfall einfach nicht geeignet sind. In der Abbildung 2.10 sind einige prominente Vertreter aufgeführt.

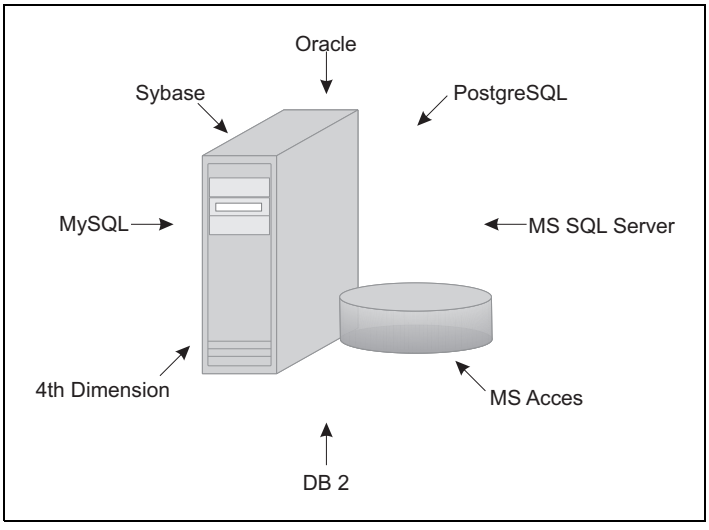


Abbildung 2.10: Produkte verschiedener Hersteller

Die bereits erwähnten Schlüsselkriterien sollen in den nächsten Abschnitten genannt und genauer beschrieben werden.

2.3.1 Leistungsfähigkeit

Datenbankhersteller werben gern mit raschen Zugriffsmöglichkeiten. Es muss an dieser Stelle sicherlich nicht darüber diskutiert werden, dass Geschwindigkeit sehr relativ ist. Geschwindigkeit bei einer Datenbankanwendung ist abhängig von der Konfiguration des Gesamtsystems, also Hardware, Betriebssystem, Netzwerk und nicht zuletzt von der Anwendung selbst. Wenn ein Anbieter mit guten oder sogar besten Zahlen bei Zugriffszeiten wirbt, dann heißt dies noch lange nicht, dass die Anwendung automatisch durch hohe Geschwindigkeit besticht. Um jedoch einigermaßen zuverlässige und vergleichbare Werte für die Leistungsmessung zu bekommen, bedient man sich so genannter *Standard-Benchmarks*. Allgemein handelt es sich bei einem Benchmark (Bezugspunkt) um ein standardisiertes Programm, mit welchem das Leistungsverhalten eines Hard- oder Softwaresystems gemessen und bewertet werden kann. Bei diesen Benchmarks werden Systemaspekte wie Prozessorleistung und Speicherausbau

berücksichtigt. Als Kenngröße wird bei DBMS, oftmals TpM (Transaktionen pro Minute) angegeben.

2.3.2 Funktionalität

Mit Funktionalität ist insbesondere die Ausstattung des DBMS mit zusätzlichen Werkzeugen, die über das einfache Ein- und Ausgeben von Daten hinausgeht, gemeint. Man ist vielleicht versucht, hierbei zuerst an moderne Schlagworte, wie Data Warehousing, OLAP und XML-Unterstützung zu denken. Dem Datenbankprogrammierer wird beim täglichen Umgang mit SQL-Skripten eine ausgereifte Programmier- und Testumgebung für diese auch gut gefallen.

2.3.3 Kompatibilität

Eine Datenbank muss austauschbar sein, d.h. bei steigenden Anforderungen an das DBMS soll es möglich sein, problemlos auf höhere Versionen des gleichen Herstellers oder aber auch auf Fremdsysteme umzusteigen. Dabei sollen Anwendungen, die auf die Datenbank zugreifen, unverändert weiter arbeiten. Auch abwärts sollte Versionskompatibilität gewährleistet sein, damit ein Datenaustausch auf keine größeren Hindernisse stößt. Eine große Anzahl von Export- und Importformaten erleichtert ebenfalls das Zusammenarbeiten mit Fremdsystemen und damit auch das Zusammenarbeiten mit Geschäftspartnern.

2.3.4 Sicherheit

Die Sicherheit ist ein Aspekt, der nicht erst im produktiven Betrieb berücksichtigt werden sollte. Der Großteil der Firmendaten ist hoch sensibel und sollte deshalb nur für einen bestimmten Personenkreis einsehbar sein. Außerdem sollen die Daten permanent verfügbar sein. Systemstörungen dürfen keinen negativen Einfluss auf die Daten selbst haben. Diese Fakten erfordern vom DBMS entsprechende Sicherheitsmechanismen. Durch Lastverteilung auf mehrere DB-Server (Clustering) kann beim Ausfall eines oder mehrerer Server der Betrieb trotzdem aufrechterhalten werden. Für Hochverfügbarkeitssysteme ist diese, vom DBMS gegebene, Möglichkeit unerlässlich.

2.3.5 Administration

Unter Administration versteht man in diesem Zusammenhang das direkte Arbeiten an und mit der Datenbank. Bei steigender Funktionalität steigt auch die Anforderung an den Administrator. Ein sehr komplexes System kann durch Kosten, die beim Schulen des Personals anfallen, uninteressant werden, obwohl hohe Funktionalität eigentlich für dieses System sprechen würde. Außerdem gilt es zu bedenken, dass unter Umständen Experten die Installation des Systems durchführen müssen. Für Systeme, die über

mehrere Standorte verteilt sind, ist es von Vorteil, wenn die Administration über Fernzugriff möglich ist. Das erlaubt eine zentralisierte Wartung und reduziert Personalkosten.

2.3.6 Preis

Der Preis ist neben den technischen Anforderungen und im Vergleich mit diesen, sicherlich das wichtigste Kriterium, welches zur Kaufentscheidung herangezogen wird. Die Kosten reduzieren sich, wie bei den vorherigen Punkten bereits erwähnt, nicht nur auf einmalige Anschaffungskosten des Systems. Auch Schulungen, zusätzliche Hardware, weiteres Personal oder abenteuerliche Lizenzpolitik des DB-Herstellers können zu unerwarteten Zusatzkosten führen. Schließlich können je nach Hersteller weitere Serviceleistungen wie Support richtig ins Geld gehen. Doch selbst die Anschaffungskosten sind bisweilen recht schwierig zu ermitteln. Der schließlich zu zahlende Endbetrag setzt sich aus mehreren Größen wie Anzahl der Benutzer, Datenträger, Rechnerplattform für Client und Server, Runtime- oder zusätzliche Entwicklungsumgebung zusammen. Weitere Optionen lassen sich gegen Aufpreis käuflich erwerben. Deshalb ist ein echter Preisvergleich recht schwierig manchmal nahezu unmöglich.

2.3.7 Fazit

Vor der Entscheidung für oder gegen ein DBMS muss ein möglichst genaues Anforderungsprofil erstellt werden. Es ist nicht erforderlich, große Geschütze aufzufahren, wenn es darum geht, kleine Probleme aus der Welt zu schaffen. Ein Gemüsehändler, der seine 40 Lieferantenadressen computerunterstützt erfassen will, benötigt dafür nicht ein DBMS von Oracle oder IBM. Für Anwendungen mit eher geringen Anforderungen kann man mit dem Freeware Produkt MySQL oder mit MS Access schnell und kostengünstig zu Lösungen kommen. Besonders der Einsteiger, der noch wenig Kenntnisse hat, kann über die grafische Benutzeroberfläche von MS Access schnell Fortschritte machen.

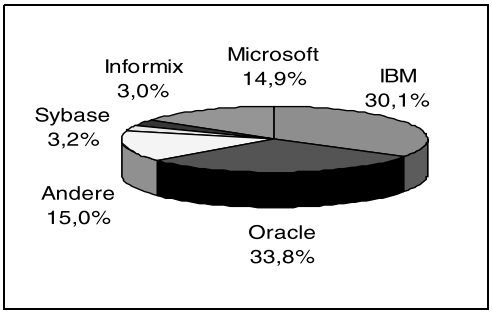


Abbildung 2.11: Marktaufteilung der DBMS-Hersteller für das Jahr 2000 [Garduer Group]

MySQL und MS Access fehlen jedoch einige grundlegende Funktionen, die ein DBMS auszeichnet und werden deshalb nicht zu diesen gezählt. Als Datenquelle für kleinere Anwendungen spielen sie aber durchaus eine Rolle.

Wie die Grafik zur Marktverteilung zeigt, sind die Produkte von Oracle und IBM ungefähr gleich auf. Berücksichtigt sind hier die Umsatzzahlen der Hersteller. DB2 von IBM hat laut Analysten eindeutige Vorteile beim Lizenzmodell und den Betriebskosten, wohingegen Oracle mit ausgereifteren Zusatzwerkzeugen aufwarten kann. Bei den Werten bzgl. der Leistungsfähigkeit stehen diese beiden auch an der Spitze.

Der SQL Server von Microsoft belegt bei den Umsatzzahlen einen Mittelfeldplatz, was jedoch nicht heißt, dass er für Highend-Lösungen völlig untauglich ist. Die Vorteile des SQL Servers liegen jedoch eindeutig in seiner relativ einfachen Handhabung bei einer gleichzeitig komfortablen Ausstattung an Hilfswerkzeugen.

PostgreSQL ist von den aufgeführten DBMS die einzige kostenlose Lösung. Im Anhang finden Sie die Internetadressen über die Sie das System beziehen können. Weitere Beispiele können somit direkt in der Praxis überprüft werden.

2.4 Datenbankarchitektur

Um einen Affen zu füttern, muss man nicht wissen, wie es um seine Psyche bestellt ist. Vielleicht ist es aber doch interessant zu erfahren, warum er das eine mal die Bananen bereits aus der Hand klaut und das andere mal die Nüsse angewidert zurückwirft.

Beim DBMS ist es ebenfalls nicht erforderlich die Architektur zu kennen, wenn man sich damit begnügt, das System mit Daten zu füttern oder sie abzurufen. Will man jedoch mehr über die Verhaltensweise des DBMS, gerade auch bezüglich der Anwendungsentwicklung, erfahren, ist es nützlich, die Arbeitsweise der Datenbank-Engine zu kennen.

Zunächst sollen allgemeine Strukturen vorgestellt werden, um dann später darauf aufbauend die Architekturen für die in diesem Buch behandelten DBMS vorzustellen.

2.4.1 Allgemeine Datenbankarchitektur

Bei der Beschreibung eines RDBMS haben wir bereits gelernt, dass es mindestens zwei Ebenen gibt, auf denen Aktionen der Datenbankanwendung stattfinden. Die für den Benutzer sichtbare und zugängliche Ebene, sowie eine Ebene auf der programmintern die Daten abgerufen werden. Diese für den Benutzer unzugängliche Ebene soll nun nochmals aufgeteilt werden in eine konzeptuelle Ebene, welche die logische Sicht auf die Daten darstellt und eine interne Ebene, bei der die physikalische Beschreibung erfolgt. Im Modell nach ANSI/SPARC sind diese drei Ebenen der Datenbankarchitektur in einer Grafik zusammengefasst.

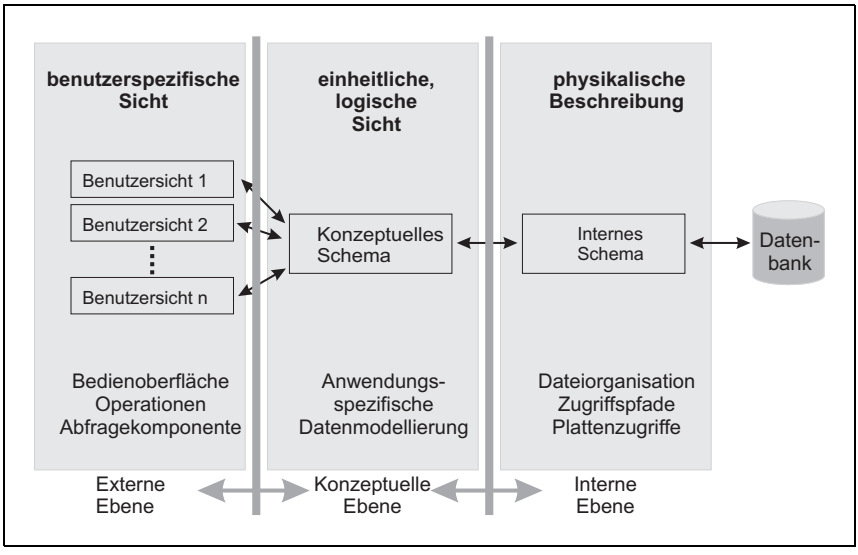


Abbildung 2.12: Datenbankarchitektur nach ANSI/SPARC

Die externe Ebene wird in Kapitel 6 beschrieben, wenn es darum geht, Richtlinien für die Gestaltung der Oberfläche zu erörtern. Zum konzeptuellen Schema kommen wir, wenn es um den Datenbankentwurf bzw. um die Datenbankabfragesprache SQL geht (Kapitel 3 und 4).

Ein grundsätzliches Schema für die interne Ebene eines RDBMS zeigt die Abbildung 2.13.

Hier werden die verschieden Schritte beschrieben, die intern ablaufen, wenn eine SQL-Abfrage vom DBMS bearbeitet wird. In diesem Zusammenhang sollen einige grundlegende Module benannt werden.

Zunächst wird im *Syntaxanalysierer* der SQL-Befehl auf seine syntaktische Korrektheit hin überprüft. Die Abfrage wird anschließend zurückgeschrieben und mittels Systemdaten und Definitionstabellen optimiert. Es wird ein Plan erstellt, der das schnellste Verfahren für den Datenzugriff erhält. Dafür werden z.B. die Datenmenge in den Tabellen, Existenz und Struktur von Indizes, Vergleichsoperatoren und Schlüsselwörter der SQL-Anweisung als Kriterien herangezogen.

In der *Ausführungseinheit* wird der Ausführungsplan gestartet. Das Modul durchläuft alle Befehlsschritte des Ausführungsplans in Schleifen, bis der Stapel abgearbeitet ist. Die meisten Befehle erfordern die Zusammenarbeit mit dem Puffer-Manager, um Daten zu verändern oder zu ermitteln, sowie um Transaktionen und Sperren zu verwalten.

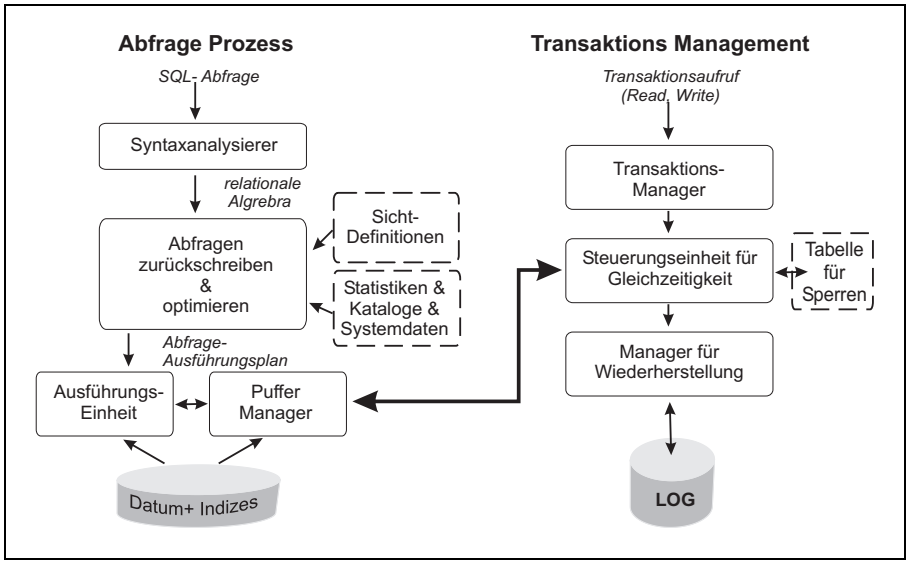


Abbildung 2.13: Interne Ebene eines RDBMS

Eine Transaktion ist ein Stapel an SQL-Anweisungen, der als Ganzes behandelt und abgearbeitet wird (näheres zu Transaktionen erfahren Sie in Kapitel 8). Wollen mehrere Benutzer gleichzeitig auf einen Datensatz zugreifen, gibt es Probleme beim Modifizieren dieses Datensatzes. Diese Problematik wird bei der Steuerungseinheit für Gleichzeitigkeit berücksichtigt. Es werden Ressourcen zugewiesen bzw. gesperrt. Wann welcher Benutzer auf welche Ressourcen zugreifen darf, ist in Systemtabellen beschrieben und kann durch entsprechende Programmanweisungen neu definiert werden. Wird eine Transaktion nicht korrekt ausgeführt, so wird die Datenbank wieder in den Zustand versetzt, in dem sie sich vor Verarbeitung des Stapels befand. Außerdem werden alle Änderungen im Transaktionsprotokoll (Log-Datei) festgehalten.

Der *Puffer-Manager* verwaltet die im Speicher befindlichen Versionen aller physischen Festplattenseiten und bietet allen anderen Modulen (bei Berücksichtigung der Sicherheitsmaßnahmen) Zugriff darauf. Wenn eine Seite für einen Prozess benötigt wird, muss sie im Arbeitsspeicher vorliegen. Befindet sie sich nicht dort, wird ein Festplattenzugriff ausgeführt, um sie zu holen. Der Zugriff auf die Festplatte ist mit erhöhtem Zeitaufwand verbunden, deshalb ist ein größerer Arbeitsspeicher, der mehr Datenseiten beinhalten kann, von Vorteil. Die Wahrscheinlichkeit, die benötigte Datenseite dann im Arbeitsspeicher vorzufinden, ist entsprechend größer.

Der Puffer-Manager reagiert mit einem Zeiger auf den Speicher, in dem die Datenseite steht. Die Reaktion kann sofort erfolgen, wenn die Seite bereits im Arbeitsspeicher ist, oder es dauert einen Augenblick, bis ein Festplattenzugriff erfolgt ist. Bei einer Aktua-

lisierung der Datenseite ist auch der Puffer-Manager dafür zuständig, dass diese auf die Festplatte zurückgeschrieben wird und dabei mit Sperren- und Transaktionsverwaltung koordiniert.

2.4.2 MS SQL Server

Die Abbildung zeigt die vereinfachte Engine des SQL Servers, in der die Komponenten in verschiedenen Schichten angeordnet sind. Die oberste Abstraktionsschicht ist die Netzwerkbibliotheksschicht. Durch die Netzwerkbibliothek (oder auch Net-Library) ist der SQL Server in der Lage, im Netzwerk mit anderen Rechnern zu kommunizieren.

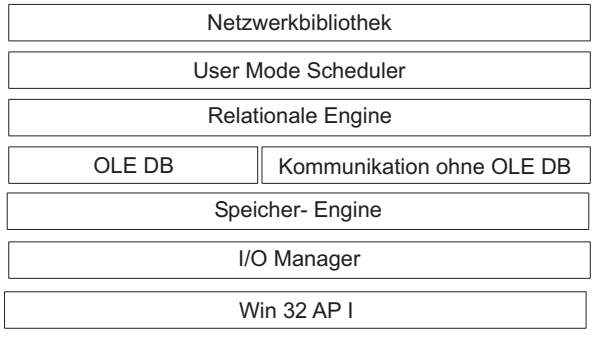


Abbildung 2.14: Architektur des MS SQL Servers

Auf der nächsten Ebene besteht die Möglichkeit, die Windows-Authentifizierung auch für den SQL Server zu nutzen, um eine vertraute Verbindung aufzubauen. Oder es soll eine spezielles Anmelde-ID/Kennwort-Paar für die Berechtigung der Benutzer festgelegt werden.

Die Relationale Engine beinhaltet weitere Komponenten wie ODS (Open Data Services), den Parser, TSQL-Compiler und Optimierer.

Die *ODS* verwalten das Netzwerk. Der Aufbau von neuen Verbindungen oder das Fehlschlagen von Verbindungen wird kontrolliert bzw. Statusmeldungen werden an den Client zurückgegeben.

Der *Parser* überprüft abgesetzte SQL-Anweisungen auf korrekte Syntax und übersetzt Transact-SQL-Befehle in ein internes Format. Dieses interne Format wird auch *Abfragebaum* genannt.

Der *Abfrageoptimierer* übernimmt den Abfragebaum vom Parser und bereitet ihn für die Ausführung vor. Dieses Modul kompiliert einen vollständigen Befehlsstapel, optimiert Abfragen und prüft die Sicherheit. Aus Optimierung und Kompilierung der Abfrage ergibt sich ein *Ausführungsplan*.

Die Relationale Engine verwendet überwiegend OLE DB (Object Linking and Embedding) zur Kommunikation mit der Speicher-Engine. In der Speicher-Engine befinden sich Module, die das Ablegen und Abrufen von Daten steuern. So verwaltet der Index-Manager beispielsweise Suchoperationen, die auf Indizes zurückgreifen. Weitere Module sind Zeilenoperations-Manager, Sperren-Manager, Sortier-Manager, Protokoll-Manager und diverse Dienstprogramme.

Über den *I/O Manager* und der Schnittstelle Win32 API erfolgen die Anforderungen an das Betriebssystem. Da der SQL Server ein Microsoft Produkt ist, werden hier insbesondere Funktionen von Microsoft Betriebssystemen genutzt.

2.4.3 PostgreSQL

Im Datenbanken-Jargon, benutzt PostgreSQL ein einfaches »Process Per-User« Client/Server-Modell. Eine PostgreSQL-Sitzung besteht aus den folgenden Prozessen (Programmen):

- Ein Supervisor-Domain Prozess (postmaster),
- Die Benutzer Frontend-Anwendung (z.B. das psql Programm),
- Ein, oder mehrere, Datenbankserver (der Postgres-Prozess selber).

Ein einziger Postmaster verwaltet eine gegebene Sammlung von Datenbanken auf einem Host. Eine solche Sammlung von Datenbanken wird Installation oder Site genannt. Frontend-Anwendungen, welche einen Zugriff auf eine Datenbank in einer Installation wünschen, führen einen Aufruf an die Bibliothek durch. Die Bibliothek sendet dann Requests (Anfragen) über das Netzwerk an den Postmaster, der sobald er an der Reihe ist, einen neuen Backend-Server-Prozess startet und den Frontend-Prozess mit dem neuen Server verbindet. Von diesem Punkt an kommunizieren der Frontend-Prozess und der Backend-Server ohne Einmischung des Postmasters. Deshalb läuft der Postmaster immer, um auf Requests zu warten, die von Frontend- und Backend-Prozessen kommen und gehen.

Die libpq-Bibliothek erlaubt einer einzelnen Frontend-Anwendung mehrere Verbindungen zu Backend-Prozessen aufzubauen. Jedoch ist die Frontend-Anwendung immer noch ein single-thread-Prozess. Multithread Frontend/Backend-Verbindungen werden von libpq nicht unterstützt. Der Postmaster und der Backend-Prozess laufen immer auf dem gleichen Rechner (dem Datenbankserver), während die Frontend-Anwendung auf einem anderen Rechner läuft. Sie sollten das beachten, weil die Dateien, auf die von einem Client-Rechner zugegriffen werden kann, eventuell vom Datenbankserver aus nicht mehr verfügbar sind.

2.4.4 MS Access

Die verschiedenen Komponenten, die in Access zur Verfügung stehen, finden wir in Abbildung 2.15.

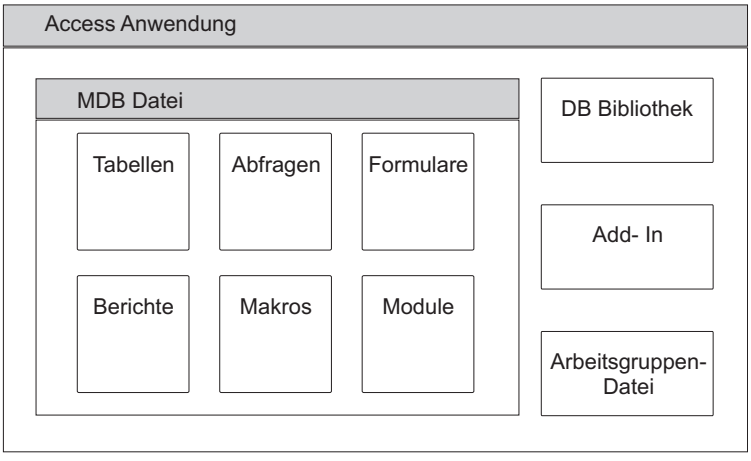


Abbildung 2.15: Komponenten in Access

Die Tabellen entsprechen der Struktur, die wir bereits beim relationalen Modell kennen gelernt haben. Eine Tabelle ist in den meisten Fällen einer Access-Anwendung die logische Datenquelle. Aus der Tabelle werden über Verknüpfungen und Abfragen Informationen für die Berichte geliefert. Der Benutzer kann über Eingabeformulare Daten in die Tabellen einfügen. Zu diesem Zweck müssen diese vorher mit der Tabelle verknüpft sein oder die Daten werden mittels Datenzugriffsprozeduren, welche in den Modulen als Programmcode erstellt werden, aus den Formularen gelesen oder zurückgeschrieben. Es gibt bereits einige Funktionen und Prozeduren, die der Anwendung aus den DB-Bibliotheken oder als Add-In zugefügt werden können. Anders als die Tabellen, Formulare, Abfragen, Makros, Module und Berichte sind diese kein Bestandteil der MDB-Datei. In der Arbeitsgruppeninformationsdatei (*.MDW) stehen die Sicherheitsinformationen für Access.

Nicht nur die Tabellen bzw. die MDB-Dateien können als Datenquelle dienen. Mit der OLE DB-Architektur und den ActiveX Data Objekten bemüht sich Microsoft um eine Strategie des universellen Datenzugriffs. OLE DB (Object Linking and Embedding DB) ist eine komponentenbasierte Datenbankarchitektur, die einen effizienten Netzwerk- bzw. Internetzugriff auf vielerlei Arten von Datenquellen implementiert. Zu diesen Datenquellen gehören relationale Datenbanken, E-Mail-Dateien, unstrukturierte Dateien und Dateien mit Tabellendaten. Im Rahmen der OLE DB-Architektur wird die

Anwendung, die auf die Daten zugreift, als Datenverbraucher bezeichnet, während das Programm, das den Datenzugriff ermöglicht, als Datenbankanbieter bezeichnet wird.

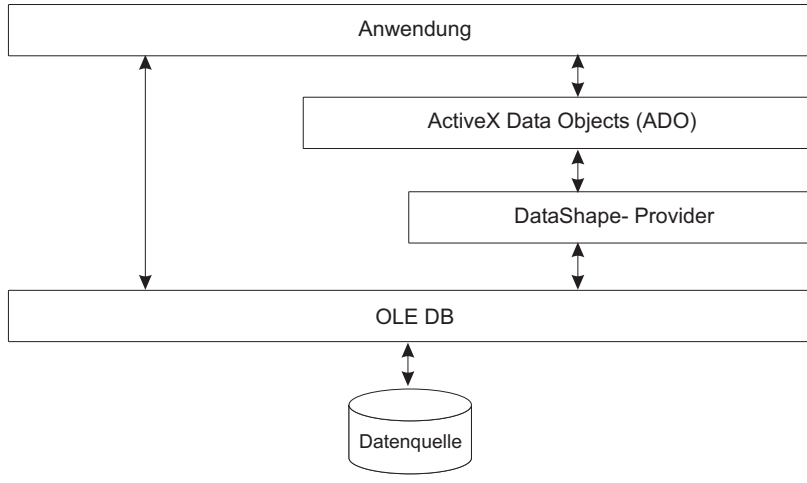


Abbildung 2.16: Datenzugriff auf eine Datenquelle mit OLE DB und ADO

ActiveX Data Objects (ADO) ist die Bezeichnung für Objekte, durch die ein Zugriff auf Datenbanken und andere Datenquellen ermöglicht wird. Diese Objekte basieren auf OLE DB, welches die Ansteuerung der Datenquelle übernimmt.

2.5 Programmiersprachen für die Datenbankapplikation

Mit Programmiersprachen besitzt der Mensch eine Kommunikationsmöglichkeit mit der Maschine. So waren die ersten Programmiersprachen auch sehr maschinenorientiert, d.h. die Sprache kam eher den technischen Gegebenheiten, als dem menschlichen Sprachvermögen entgegen. Höhere oder »problemorientierte« Programmiersprachen sind maschinen- bzw. prozessorunabhängig und bestehen aus Anweisungen. Sie müssen vor der Ausführung von einem Übersetzer (Compiler) in die Maschinensprache übersetzt werden. Seitdem, Mitte der 60er Jahre, mit Fortran die erste höhere Programmiersprache entstand, wurden hunderte weitere Sprachen entwickelt (Abbildung 2.17).

Universalsprachen sind für ein breites Anwendungsgebiet, wie etwa technisch-wissenschaftliche Probleme, konzipiert.

Stapelorientierte Sprachen basieren auf dem Modell der Ein- und Ausgabe von Daten. Während ein Algorithmus durchlaufen wird, hat der Benutzer weitestgehend keine Möglichkeit der Einflussnahme.

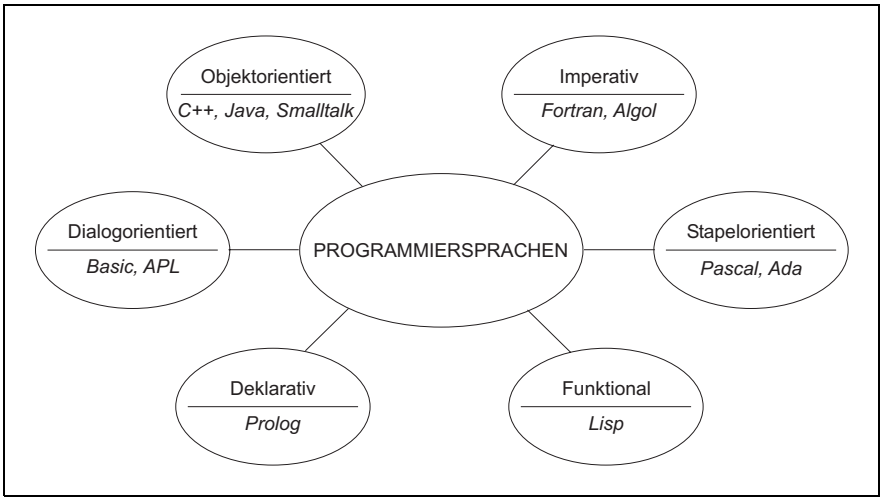


Abbildung 2.17: Eigenschaften von Programmiersprachen mit Beispielen

Dialogorientierte Sprachen hingegen bieten die Möglichkeit der Interaktion zwischen Mensch und Maschine. Ein Programm kann durch den Benutzer gesteuert und stückweise übersetzt werden.

Imperative Sprachen bestehen aus einer Folge von Anweisungen. Es werden Variablen im Sinne von Behältern (Speicherplätzen) verwendet, in die man zeitabhängig verschiedene Datenwerte legen kann.

Funktionale Sprachen beruhen auf dem Modell der mathematischen Funktion. Jeder Algorithmus kann als Funktion verstanden werden, die Argumente in Ergebnisse abbildet.

Deklarative Sprachen beschreiben Daten und Beziehungen zwischen ihnen. Der Algorithmus ist in der Semantik der Sprache verborgen. Sie haben daher einen hohen Abstraktionsgrad.

Objektorientierte Sprachen beruhen auf dem Prinzip, dass alle Objekte (Daten) selbstständig und aktiv sind. Die Objekte können Prozeduren ausführen und Nachrichten an andere Objekte senden.

Visuelles Programmieren nimmt einem weitestgehend die Arbeit für die Erstellung der Oberfläche ab. Nach dem Baukastensystem werden Fenster, Menüs sowie weitere Standardelemente ausgewählt und mit Programmiercodes für die Ereignisbearbeitung versehen. Zuerst war dies bei Visual Basic Standard, gefolgt von Delphi, C++ und Java.

Aufgrund verschiedener Motivationen und unterschiedlicher technischer Anforderungen gibt und gab es also verschiedene Programmiersprachen, die sich sowohl in der Syntax als auch in der Semantik stark unterscheiden. Welche Sprache eignet sich also

für die Datenbankanwendung? Diese Frage stellt sich natürlich insbesondere dem Datenbankprogrammierer. An dieser Stelle sollen nun vier Sprachen etwas ausführlicher beschrieben werden, die alle Datenbankfunktionen zur Verfügung stellen.

Visual Basic

Basic wird auch heute noch als die typische Einsteigersprache gehandelt. Es war in den 80er Jahren in Heim- und Taschencomputern integriert und war für so manchen Schüler oder Studenten ein Start in die Programmierwelt. Heute bietet Visual Basic, im Gegensatz zu QBasic oder Power Basic (als Basic noch in den Kinderschuhen steckte), eine enorme Leistungsfähigkeit. Elemente aus dem objektorientierten Modell und Netzwerkbibliotheken gehören ebenso zum Standard wie moderne Datenzugriffsobjekte und Unterstützung von Internetprotokollen. Gleichzeitig ist eine gewisse Leichtigkeit bei der Programmierung erhalten geblieben. Wesentliche Neuerungen sind:

1. Module – Der Modulbegriff deckt sich nicht mehr mit dem Programmbegriff, sondern erweitert die Strukturierungsfähigkeit der Sprache nach oben hin. Die Sprache vermag damit insbesondere Bibliotheken (Klassen- und Komponentenbibliotheken) konzeptuell zu integrieren.
2. Objekte – Der Objektbegriff ist inzwischen sprachlich-syntaktisch verwurzelt. Neben den einfachen Datentypen stehen nun auch komplexe, verkapselte Datentypen samt zugehöriger Operationen zur Disposition. Der Sprache eröffnet sich damit unter anderem die vollständige Anbindung an alle Dienste des Betriebsystems inklusive der grafisch orientierten Benutzerschnittstelle, des gesamten Dateiwesens, der Kommunikationseinrichtungen und der Datenbankschnittstellen.
3. Ereignisse – Die Sprache adaptiert das Modell der ereignisgesteuerten Anwendungspro-grammierung von Windows über den Objektbegriff. So gestatten vordefinierte Klassen und interaktiv definierte Objekte dieser Klassen eine umfassende Repräsentation der Benutzerschnittstelle mit all ihren Elementen (Fenster, Menüs, Symbol- und Statusleisten, Steuerelemente, Komponenten) und Aktionen.
4. ActiveX – Die vom COM geforderte Codeabstraktion ermöglicht es der Sprache nicht nur, von existierenden Komponenten zu profitieren, sondern auch selbst Komponenten beizusteuern, von denen wiederum andere Anwendungen profitieren können.

Ein weiterer Grund, der für Visual Basic spricht, ist die weite Verbreitung von VBA (Visual Basic für Applikationen). VBA ist als Makrosprache in Standardanwendungen wie MS Word oder MS Access zu finden. In VBA gibt es viele elementare Sprachkonstrukte aus Visual Basic und zusätzlich anwendungsspezifische Funktionen. Bereits innerhalb der Anwendungsumgebung besitzen Sie mit VBA oftmals eine recht komfortable Entwicklungsumgebung mit Codeeditor, Debugger und Objektkatalog.

Pascal

Pascal wurde zuerst wie Basic als Interpreter-Sprache konzipiert, mit dem Unterschied, dass Basic während des Ausführens übersetzt wird, während Pascal zuerst ein Zwischencode bildet, der dann zur Laufzeit übersetzt wird. Der Vorteil: Der Zwischencode war maschinenunabhängig und nur die Ausführung zur Laufzeit musste an die Maschine angepasst werden. Trotzdem ist dies sicher einer der Gründe, warum heute Pascal nicht *die* Rolle spielt, denn richtig schnelle Programme kann man damit im Gegensatz zu Sprachen, die von vornherein als Compilersprache ausgelegt wurden, nicht erzeugen.

1982 brachte Borland einen Pascal Compiler für CP/M und DOS namens Turbo Pascal auf den Markt, der Maßstäbe setzte – schnell im Übersetzen, schnelle ausführbare Programme, bequemes Erstellen im integrierten Editor.

Borland entwickelte Pascal schließlich unter der Bezeichnung *Delphi* objektorientiert weiter. Zwar setzen viele Entwickler Delphi zuhause ein, aber kommerziell spielt es nicht eine so große Rolle wie etwa Visual Basic. Bei der Vorstellung von Delphi wurde großer Wert auf die Datenbankfähigkeiten gelegt. Eventuell wurde so bei manchem das Gefühl vermittelt, dass es sich eher um einen Datenbank-Frontend, als um eine universelle Programmiersprache handelt. Ein Vorteil von Delphi sind die verfügbaren Komponenten (AddIns), also Bausteine, die man in Programme einbinden kann und dadurch Funktionen hat, die Delphi nicht mitbringt.

C++

Auch C gibt es inzwischen objektorientiert und visuell. Die Weiterentwicklung läuft nun unter der Bezeichnung C++, was in C soviel heißt wie »C um eins erhöht«. Oftmals wird ein kombinierter C/C++ Compiler angeboten. In Wirklichkeit handelt es sich jedoch um eine mächtige neue Programmiersprache, bei der man zum einen die Mängel von C hinsichtlich Sicherheit beseitigt und zum andern einen objektorientierten Ansatz hat. C++ ist von den Sprachmöglichkeiten die umfangreichste der hier vorgestellten Sprachen. Dadurch ist sie aber auch sehr komplex und nicht so einfach erlernbar.

C++ entstand ab 1982 aus C. Zunächst als Klassenerweiterung mit Anleihen an *Simula*. C++ führte neben den Objekten auch weitere Spracherweiterungen ein, wie die Fehlerbehandlung oder die Möglichkeit, selbst Operatoren zu definieren. C++ ist nahezu vollständig kompatibel zu C, das heißt, jedes C-Programm kann auch als C++ Programm übersetzt werden.

C++ ist heute in der Anwendungsentwicklung von allen hier vorgestellten Sprachen die wichtigste auf dem Gebiet der hardwarenahen Programmierung. Dafür ist die Auswahl an Compilern sehr groß. Nicht nur für Windows, für nahezu jedes Betriebssystem gibt es C++ Compiler.

Java

Zwar ist es möglich, mit C++ Programme für verschiedene Plattformen zu erstellen, aber durch die verschiedenen Dialekte, die auch durch diverse Compiler-Anbieter begründet sind, ist die Maschinenunabhängigkeit verlorengegangen.

Mit Java ging die Firma Sun als Hardwarehersteller einen anderen Weg. Anstatt die Sprache jeweils der Maschine anzupassen, entwickelte man eine Sprache für alle Maschinen, deren verschiedene Funktionen in Klassen abgelegt wurden. Daraus erzeugt der Java Compiler einen maschinenunabhängigen Code, der von einem Interpreter jeweils maschinenspezifisch ausgeführt wird. Dieser Interpreter – die *Java Virtual Maschine* – ist dem Betriebssystem angepasst und wird auf dem PC installiert. Wie der Name vermuten lässt, ist es nichts anderes als ein Emulator eines »Java PCs«.

Der Vorteil ist, dass ein Java-Programm prinzipiell auf jedem Rechner läuft, und man den Zwischencode nur einmal erstellen muss. Auf jedem Rechner muss nur ein Interpreter installiert sein, der Java ausführt.

Damit war Java die Sprache für das Internet. Egal, ob man mit einem Macintosh, einem Windows- oder Unix-Rechner oder einer Sun Workstation ein Java Programm abruft, es sollte immer laufen. Weiterhin gilt Java als sehr sicher. Java-Anwendungen haben keinen Zugriff auf das Dateisystem des Rechners, auf dem sie laufen.

Sowohl die kostenlose Weitergabe der Java-Entwicklungsumgebung »*Java Development Kit*« (JDK) als auch die Implementation der »Java Virtual Machine« in verschiedene Web-Browser, fördern die Verbreitung von Java in beträchtlichem Maße. Durch zusätzliche Standardfunktionalitäten wie *JDBC* (Java Database Connectivity) wird Java zu einer der mächtigsten Programmiersprachen für verschiedene Anwendungsgebiete. Insbesondere können Seiten des World Wide Web zu echten Client/Server-Anwendungen werden. Das im Browser ablaufende Programm erzeugt eine Anfrage an die Datenbank, erhält das Ergebnis und präsentiert dieses in Form einer www-Seite.

Die Programmbeispiele, die Sie in diesem Buch finden, werden entweder in Visual Basic oder aber in Java vorliegen.

2.6 Normung und Qualitätssicherung

»Softwareentwicklung ist ein kooperatives Spiel, in dem die Mitspieler Markierungen und Requisiten verwenden, um sich gegenseitig und selbst zu erinnern und über den nächsten Spielzug zu informieren und zu inspirieren.«
Alistair Cockburns – Manifest der Softwareentwicklung

2.6.1 Normung

So mancher sieht beim Begriff *Normung* seine künstlerische Freiheit in Gefahr. Individualität in der Kunst ist eine großartige Sache, aber stellen Sie sich vor, Sie wollen Ihren Arbeitsspeicher erweitern und jeder Hersteller hat für die Chips eine andere PIN-Belegung gewählt. Oder nehmen wir ein Beispiel aus dem Bankwesen. Wie wollen Sie sich in einer fremden Stadt oder einem fremden Land Geld vom Automaten auszahlen lassen, wenn jedes Kreditinstitut sein eigenes Format für die Geldkarte hat? Mittlerweile haben wir quer durch die verschiedenen Anwendungsbereiche die Arbeit der Normungsinstitute zu schätzen gelernt. Doch was verbirgt sich überhaupt hinter dem Begriff »Normung«? Der Wortsinn hat je nach Verwendungsbereich eine andere Färbung. Deshalb sollen die, die sich damit beschäftigen, also das Deutsche Institut für Normung e.V. (DIN), eine Definition liefern:

»Normung ist die planmäßige, durch die interessierten Kreise gemeinschaftlich durchgeführte Vereinheitlichung von materiellen und immateriellen Gegenständen zum Nutzen der Allgemeinheit.«
DIN 820

Normung begünstigt durch seine Vereinheitlichung und Formalisierung die Rationalisierung und Qualitätssicherung. Das scheint zunächst nur ein Vorteil für den Arbeitgeber zu sein, doch wenn man die gesamte Bandbreite der Normung zur Kenntnis nimmt, erkennt man die Vorteile für alle Lebensbereiche. So schützen Normen im Verbraucher-, Arbeits-, Unfall- und Umweltschutz zunächst den Menschen vor den Folgen der Technik.

Welche Normungsgremien gibt es und wo sind sie geographisch anzuordnen?

Allgemein bekannt ist das bereits genannte Deutsche Institut für Normung (DIN), welches für die Verabschiedung nationaler Normen zuständig ist. Für internationale Normierung ist *International Standards Organisation (ISO)* die maßgebende Instanz.

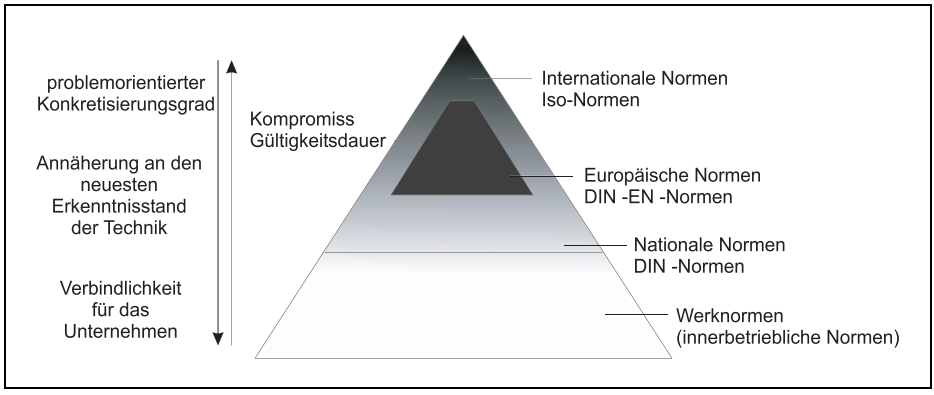


Abbildung 2.18: Normungspyramide

Internationale Organisation für Normierung (ISO)

ISO ist eine weltweite Vereinigung von über 100 nationalen Standardisierungsinstituten. 1947 wurde es gegründet, um die internationalen Standards zu entwickeln, die den Austausch von Waren und Dienstleistungen, und Mitarbeit im intellektuellen, technologischen, wissenschaftlichen und Wirtschaftsleben verbessern und erleichtern sollte. Die Hauptverantwortlichkeit von ISO wird in einer Hierarchie von 2.700 technischen Ausschüssen, von Unterausschüssen und von Arbeitsgruppen mit mehr als 20.000 Ausschussmitgliedern weltweit dezentralisiert. Die nationalen Standardkörper, welche die ISO-Mitgliedschaft – AFNOR, ABSI, BSI, CSBTS, DIN, SIS, usw. bilden, akzeptieren die Hauptverantwortlichkeit des Normungsausschusses. Das zentrale Sekretariat befindet sich in Genf. Von dort wird der Fluss der Unterlagen in allen Richtungen sichergestellt. Über 9.000 internationale Standards sind bisher veröffentlicht worden. ISO behält nahe Arbeitsrelationen mit den Normungsinstituten des einzelnen Landes bei. In der Praxis sind die Mitglieder solcher Gruppen auch Mitglieder von ISO.

Normalerweise werden ISO-Normen als die Grundlage genommen, auf denen Standards aufsetzen, die bestimmte regionale Bedürfnisse erfüllen sollen. Der Bereich von ISO wird nicht auf irgendeinen bestimmten Zweig begrenzt; er umfasst alle Normierungsfelder ausgenommen elektrische und elektronische Technik, welche in die Verantwortlichkeit von IEC gehören. Die Arbeit auf dem Gebiet der Informationstechnologie geschieht durch einen Zusammenschluss des Ausschusses der Verbindung ISO/IEC.

ANSI – American National Standards Institute

Das American National Standards Institute (ANSI), gegründet 1918, ist eine private, gemeinnützige Organisation, die für die Einrichtung vieler Standards, einschließlich einer Anzahl von Datenaustausch und Terminalstandards verantwortlich ist. ANSI

fördert den Gebrauch von US-Standards international und hat die alleinigen Rechte zu den ISO- und EDI-Standards in den Vereinigten Staaten. ANSI ist der anerkannte US-Repräsentat von CCITT. Beispiel für die Arbeit von ANSI: ANSI plante den ASCII-Zeichensatz, einen allgemein verwendeten Standarddatenübertragungscode.

Einige Ausschüsse und Arbeitsgruppen, die von ANSI gebildet werden:

1. HISPP (Healthcare Informatics Standards Planning Panel) – Healthcare Informatics Standards Board,
2. HL7 Health Level 7,
3. ASC X12 – Accredited Standards Committee X12,
4. ASC X3 – NCITS National Committee for Information Technology Standards.

NCITS ist der neue Name für beglaubigte Informationstechnologie des Normungsausschusses X3. Dieser Ausschuss sitzt in Washington DC und verabschiedet Standards auf dem Gebiet der Informationstechnologie. Seine technischen Experten nehmen im Namen der US an den internationalen Aktivitäten von ISO/IECJTC1 teil. Es gibt 41 technische Ausschüsse, die Standards in den folgenden Bereichen entwickeln:

Multimedia (MPEG/JPEG), Informationssysteme, SCSI-1, geographische Informationssysteme, Speichermedien, Datenbank (einschließlich SQL3), Sicherheit, Programmiersprachen (wie C++).

Die ungeheure Fülle an Standards kann hier selbstverständlich nicht mal im Ansatz vorgestellt werden. Für den interessierten Leser gibt es im Folgenden einige Stichpunkte mit denen weitergearbeitet werden kann. Entsprechende Internetadressen zur Thematik sind im Anhang zu finden.

Weitere Normen für die Softwareentwicklung sind enthalten in:

- ISO 9000: Qualitätsmanagement- und Qualitätssicherungsnormen,
- ISO 9001: Qualitätssicherungssysteme: Modell zur Darlegung der Qualitätssicherung in Design/Entwicklung, Produktion, Montage und Kundendienst,
- ISO 9002: Qualitätssicherungssysteme: Modell zur Darlegung der Qualitätssicherung in Produktion und Montage,
- ISO 9003: Qualitätssicherungssysteme: Modell zur Darlegung der Qualitätssicherung bei der Endprüfung,
- ISO 9004: Qualitätsmanagement und Elemente eines Qualitätssicherungssystems: Leitfaden,
- ISO 9241: Anforderungen der Aufgabe und des Arbeitsplatzes,
- ISO 9241: Anforderungen des Dialoges,

- ISO 17799: Sicherheitskonzepte in der Informationstechnik,
- DIN 66200: Betrieb von Rechensystemen,
- DIN 66230: Informationsverarbeitung; Programmdokumentation,
- DIN 66234: Präsentation von Informationen,
- DIN 66285: Anwendungssoftware; Güte- und Prüfbestimmung.

Einen Verstoß gegen Normen bedeutet nicht automatisch, dass es sich um eine Rechtsverletzung und somit auch um eine strafbare Handlung handelt. Umgekehrt ist es jedoch möglich, dass sich Gesetze und Verordnungen auf Normen stützen. Damit wäre ein Zuwiderhandeln gegen entsprechende Normen ein Gesetzesverstoß.

2.6.2 Qualitätsmanagement

Wenn Qualität gesichert werden soll, muss zunächst geklärt werden, was Qualität überhaupt ist. Wenn man 100 Anwender bzgl. der Qualität einer Software befragen würde, hätte man 100 subjektive Meinungen, aber noch keine zuverlässige Aussage über die Qualität des Produktes. Auch hier hilft uns das Deutsche Institut für Normung weiter.

Laut DIN 55350 definiert sich Qualität »in der Beschaffenheit einer Einheit bezüglich ihrer Eignung, die festgelegten und vorausgesetzten Erfordernisse zu erfüllen«. Mit anderen Worten ein Antivirus-Programm, welches einen Großteil der Viren nicht zu erkennen vermag, erfüllt die vorausgesetzten Erfordernisse nicht und würde deshalb bei einer qualitativen Bewertung nicht gut abschneiden.

Es gibt einige Merkmale, über die eine Aussage bzgl. der Qualität einer Software gemacht werden kann.

Zuverlässigkeit: Es wird erwartet, dass die Software gemäß den Spezifikationen für einen vorgegebenen Zeitabschnitt ihre Funktion erfüllt.

Verfügbarkeit: In der Praxis wird die Verfügbarkeit aus dem Verhältnis der nutzbaren mittleren Betriebsdauer und der mittleren Ausfallzeit gebildet.

Sicherheit: Unerwünschte Ereignisse, wie Programm- oder Systemabsturz sind für einen vorgegebenen Zeitraum und einer programmgemäßen Handhabung zu vermeiden.

Leistung: Über das Zeitverhalten des Produktes und der Inanspruchnahme von Ressourcen kann eine Leistungsermittlung erfolgen.

Benutzerfreundlichkeit: Der Aufwand für den Benutzer, um eine gewisse Funktion zu erfüllen, sollte so gering wie möglich sein.

Wartungsfreundlichkeit: Der Aufwand für die Erkennung eines Fehlers und seine Behebung durch vorgesehenes Personal sollte so gering wie möglich sein.

Portabilität: Der mögliche Einsatz der Software in einer geänderten technischen Umgebung wird Portabilität genannt.

Der Kunde ist König. Mit diesem schlichten Satz ist andeutungsweise der Einfluss des Kunden beim Entwicklungsprozess formuliert. Das Projekt unterliegt in jeder Phase einem gewissen Spannungsfeld, das in der Abbildung 2.19 zum Ausdruck kommt.

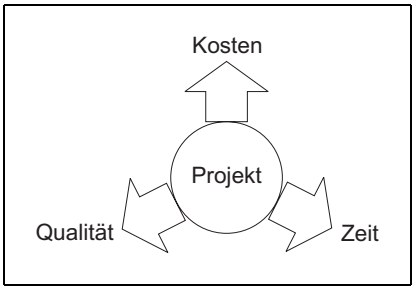


Abbildung 2.19: Spannungsfeld der Softwareentwicklung

Ganz wichtig bei der Qualitätssicherung eines Softwareproduktes ist, auch aus wirtschaftlichen Gründen, das Minimieren von Fehlern. Es ist leicht nachzuvollziehen, dass ein Fehler, der erst bei laufendem Betrieb entdeckt wird und zu einer Ausfallzeit von weiterem Personal führt, wesentlich höhere Kosten verursacht als ein Fehler, der in der Entwicklungsphase bereits erkannt und »unschädlich« gemacht wurde. Je später ein Fehler behoben wird, desto mehr Folgefehler kann er erzeugen. Der Aufwand zur Behebung des Fehlers wächst dann stärker als proportional zur Fehlerzahl.

Zur Fehlervermeidung gibt es deshalb einige Maßnahmen, die bereits während der Konstruktion angewendet werden. Man nennt sie deshalb auch *konstruktive Maßnahmen*. Dazu gehört das Verwenden moderner Entwicklungsverfahren und Testwerkzeuge. Ein modularer Aufbau des Projektes aus kleineren (bereits getesteten) Softwarebausteinen wird empfohlen. Außerdem werden Verfahren innerhalb der Software empfohlen, die eventuell auftretende Fehler abfangen und somit einen Totalausfall verhindern können.

Neben den konstruktiven Maßnahmen zur Qualitätssicherung bei Softwareprodukten, gibt es die *analytischen oder prüfenden Maßnahmen*. In diese Kategorie gehört u.a. die Durchführung einer Abnahmeprüfung für verschiedene Testfälle. Außerdem sollte es zu einer Qualitätsberichterstattung kommen, bei der die verschiedenen Entwicklungsphasen mit entsprechenden Testergebnissen festgehalten werden. Aus den Daten der Berichterstattung können grundsätzliche Bewertungen bzgl. der verwendeten Mittel

und Verfahren vorgenommen werden. So können bei einem weiteren Projekt Kosten und Werkzeuge optimiert werden.

Schließlich gibt es noch die *administrativen Maßnahmen*. Dazu gehören eine umfangreiche Dokumentationsplanung, Aufgabenverteilung bzw. Ablauforganisation, Planen von Qualitätsprüfungen und Qualitätsaudits. Auch der Hinweis auf Industrienormen und das Verfassen von Richtlinien, die im eigenen Unternehmen Gültigkeit haben, ist eine administrative Maßnahme. Damit diese Vorgaben und Richtlinien schließlich auch in den entsprechenden Abteilungen umgesetzt werden, ist es wichtig, die Mitarbeiter nicht nur zu schulen, sondern auch ausreichend zu motivieren.

Als Zusammenfassung soll die folgende Grafik dienen. Das Unternehmen bekommt vom Kunden ein Anforderungsprofil der Software. Damit der Auftrag zur Zufriedenheit des Kunden (bei gleichzeitiger ökonomischer Effizienz) erledigt wird, sollte ein System installiert werden, welches den dargestellten Kreislauf bei der Entstehung des Produktes regelt.

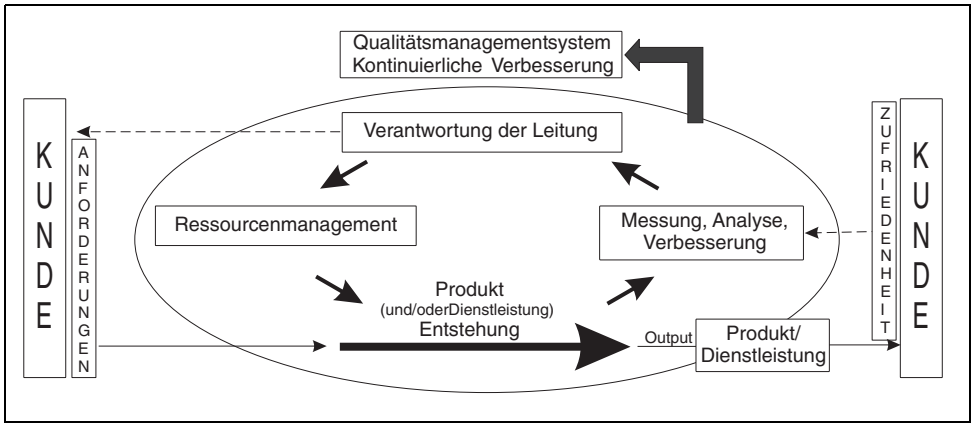


Abbildung 2.20: Qualitätsmanagement im Produktionskreislauf

Auch Programmiersprachen können einer Qualitätsüberprüfung unterworfen werden. Mögliche Qualitätsmerkmale sind hier:

- Eigenschaften bzgl. Vererbung,
- Überschreiben von Speicherbereichen durch falsch eingestellte Zeiger,
- automatische Zuordungsprüfung von Datentypen erhöht die Zuverlässigkeit,
- Konsistenz der Datenstrukturen,
- eine hohe Sprachkomplexität führt leicht zu Unübersichtlichkeit der Programme.

- Durch Software-Qualitätssicherung wird sichergestellt, dass das Produkt den Anforderungen des Kunden entspricht. Jedoch nicht nur dem Kunden dient eine gute Arbeit beim Qualitätsmanagement. Fehler in bestimmten Prozessphasen können in Folgeprojekten korrigiert werden und selbst marktstrategische Entscheidungen können durch Informationen aus dem Qualitätsmanagement beeinflusst werden.

2.7 Das Phasenmodell der Softwareentwicklung

Eine Datenbankapplikation zu erstellen ist zunächst nichts anderes als ein Programm zu entwickeln und dies wiederum ist nichts anderes als eine Problemlösung zu erarbeiten. Es gibt Programmierer, die setzten sich unmittelbar nach Beauftragung mit Tunnelblick an ihren Rechner und produzieren Programmcodes. Das ist bei komplexen Anforderungen jedoch sicherlich nicht der professionelle Weg zum Ziel, denn große Projekte lassen sich heute nicht mehr in einer »Ein-Mann-Show« realisieren. Deshalb ist hier neben einem hohen Maß an Kommunikation auch ein hohes Maß an Planung gefordert. Die Entwicklung der Software umfasst üblicherweise mehrere Phasen, von der die reine Programmiertätigkeit nur einen Teil darstellt.

Sämtliche Schritte von der Problemanalyse bis zur Wartung sind im Phasenmodell der Softwareentwicklung zusammengetragen.

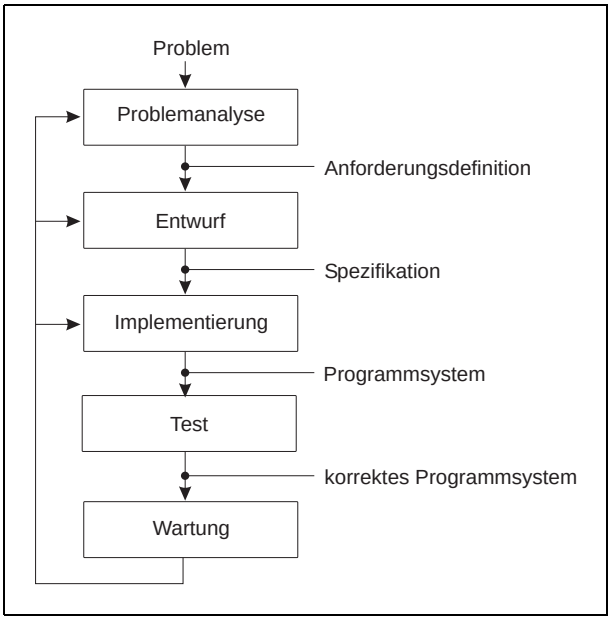


Abbildung 2.21: Phasenmodell der Softwareentwicklung

2.7.1 Problemanalyse

Auch um ungerechtfertigten Regressforderungen keinen Vorschub zu leisten, sollte zusammen mit dem Auftraggeber innerhalb einer Problemanalyse eine detaillierte Anforderungsdefinition erstellt werden. In dieser Phase ist es noch relativ leicht möglich, das Projekt aus wirtschaftlichen Gründen oder mangelnden Kapazitäten, wieder abzugeben. Deshalb müssen während dieses Projektstatus bereits Effizienzberechnungen, Bestimmung des Rechner- und Personalaufwandes und das Skizzieren eines Lösungsweges erfolgen.

2.7.2 Entwurf

Aus dem Lösungsansatz, der bei der Problemanalyse beschrieben wurde, wird beim Entwurfsprozess eine Spezifikation abgeleitet, die dann auch ein modulares Programmieren ermöglicht. Projektteile werden im Team verteilt und wenn notwendig, muss man sich um externe Ressourcen bemühen. Auch in dieser Phase sollte der Kunde »mit im Boot sitzen« und bei der Verabschiedung der Spezifikation eingebunden sein.

2.7.3 Implementierung

Nach der Spezifikation erfolgt die Implementierung, also die technische Umsetzung. Die verschiedenen Module werden nach der Spezifikation und exakt nach Schnittstellendefinition in den einzelnen Teams bzw. von den Programmierern erstellt.

2.7.4 Test

Das entwickelte Programmsystem muss in der Testphase mindestens einer syntaktischen und logischen Testprozedur zugeführt werden, wobei die Ergebnisse in Protokollen festgehalten werden sollten. Es ist ratsam, für die Protokollvordrucke firmenintern einen Standard einzuführen. Das senkt Kosten und schränkt die Existenz möglicher Fehlerquellen ein.

2.7.5 Wartung

Auch die anschließende Wartungsphase sollte frühzeitig in den Planungen berücksichtigt werden. Das Gesamtkonzept sollte so aufgebaut sein, dass die Wartung von Personal durchgeführt werden kann, das beim Entwicklungsprozess nicht beteiligt war. Eine qualitativ anspruchsvolle Dokumentation macht sich besonders in dieser Phase bezahlt. Ein Großteil der Probleme, die sich in der Wartungsphase ergeben, liegen erfahrungsgemäß an der mangelnden Schulung der Benutzer. Doch gerade bei Datenbankanwendungen kann auch Verletzung der Datenintegrität, Fragmentierung, Überlauf etc. zu Fehlern führen. Fehler also, die der Unzuverlässigkeit der Soft- und Hardware zuzuschreiben sind. Schließlich ist der Hersteller durch weitere »Begehrlichkeiten« des Kun-

den oder auch durch Veränderung der Basissoftware (DBMS, Betriebssystem, Fremdatenquellen etc.) gezwungen, die Software anzupassen. Diese Tatsache sollte ebenfalls bereits in den Entwicklungsplanungen berücksichtigt werden.

Eine eindeutige Prozessdefinition ist bei Software sehr wichtig, weil es sich um ein immaterielles Produkt handelt. Erst wenn die Prozessphasen klar definiert und dokumentiert sind, gibt es eine Basis für Diskussionen und Verbesserungen.

Testprotokoll DiLeiSoft Version: 4.02 Mittwoch, 14. November 2001

Bitte nirgendwo weitere Returns einfügen (sehr lange Zeilen sind OK)
Bei Eingabefelder die mit # gekennzeichnet sind, ist eine Angabe unbedingt erforderlich. Sämtlich JN-Fragen müssen beantwortet werden.

1.Zur Testperson
-1.01Tester_Vorname:#
-1.02Tester_Nachname:#

2.Ihre Testumgebung (z. B. Win 2000)
-2.01Test_auf_Betriebssystem:#
Haben Sie auf dem Testbetriebssystem besondere Programme
(Patches, Updates, Servicepacks, WWW-Browser) oder Hardware?
-2.02Besonderheiten_Testumgebung:
-2.03Test_etwa_begonnen_am:#

3.getestetes Programm allgemein:
-3.01Programmname:#
-3.02Versionsnr:#

4.Installation:
-4.01Installation_einfach_JN:
-4.02zur_Installation_ist_Zip_Programm_noetig_JN:
-4.03zur_Installation_muss_Installationsprogramm_gestartet_werden_JN:
-4.04zur_Installation_sind_weitere_Taetigkeiten_noetig_JN:
-4.05Dauer_der_Installation_in_Minuten_etwa:
-4.07Datum_der_installierten_Programmhaupt_Datei:

Abbildung 2.22: Beispielprotokoll für einen Programmtest

3 Datenbankentwurf

Im vorigen Kapitel wurde bereits auf die Notwendigkeit eines logischen und physikalischen Konzeptes für die Datenbank hingewiesen. Gerade unter Berücksichtigung der Aspekte, die zur Qualitätssicherheit beitragen sollen, ist der Datenbankentwurf ein wichtiger Schritt zur erfolgreichen Umsetzung des Gesamtprojektes. Fehler, die in dieser Phase gemacht werden, lassen sich später nur mühsam bereinigen. Die hier zu nennenden Aspekte sind *Vollständigkeit*, *Korrektheit*, *Minimalität*, *Lesbarkeit* und *Modifizierbarkeit*.

Beim Datenbankentwurf geht es in erster Linie darum, aus den gegebenen Anforderungen eine logische und physische Struktur für die Datenbank zu erarbeiten.

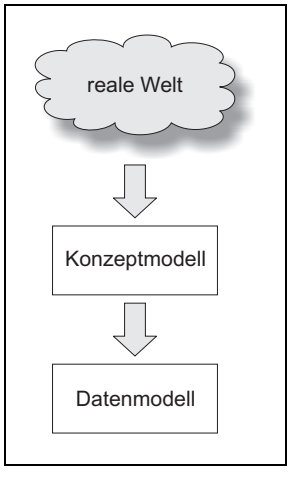


Abbildung 3.1: Von den Anforderungen der realen Welt zum Konzept

Die Normalisierung der Daten und der Entwurf mit dem Entity-Relationship Modell sind zwei Verfahren, die sich anbieten, um entsprechende Konzepte zu erarbeiten.

Die Normalisierung kann ausschließlich bei relationalen Datenmodellen verwendet werden. Das Entity-Relationship Modell hingegen ist unabhängig vom Datenmodell. Zunächst soll jedoch ein Begriff geklärt werden, der in diesem Zusammenhang eminent wichtig ist.

3.1 Ungewollte Vielfalt – Redundanz

Ein wichtiger Begriff im Zusammenhang mit Datenbanken ist der Begriff der *Redundanz*. Redundanz betrifft nicht nur Datenbanksysteme, sondern IT-Systeme im Allgemeinen. Betrachten wir z.B. Geschäftsprozesse. Gerade hier ist es wichtig, redundante Abläufe zu beseitigen, um die Laufzeiten eines Prozesses zu verringern. Bei der Programmierung, insbesondere bei der objektorientierten Programmerstellung, ist es ein wesentliches Ziel, Redundanz zu vermeiden.

Welche Bedeutung hat nun der Begriff *Redundanz*?

Übersetzt bedeutet Redundanz, dass etwas mehrmals und zwar überflüssigerweise vorliegt. Das ist zunächst nichts Verwerfliches, wenn wir an Erbformationen in unseren Zellen denken. In einer Zelle steckt die Erbinformation aller anderen Zellen. Manchmal wird auch ganz bewusst Redundanz erzeugt. In der Werbung meint man manchmal, dass das Label der Firma gar nicht oft genug im Bild erscheinen kann. Welche unangenehmen Folgen Redundanz jedoch haben kann soll das Fallbeispiel verdeutlichen.

Beispiel 3.1:

Ein Buch existiert in der Datenbank eines Buchgeschäftes zweimal als Objekt. Dem Objekt ist jeweils der Preis als Eigenschaft zugewiesen. Nun hat es einen Preisnachlass gegeben. Eine Datenbankprozedur hat dabei den Preis bei einem Objekt angepasst, jedoch nicht bei dem anderen. Ausgehend vom neuen Preis, werden die Bücher neu deklariert. Ein Kunde entscheidet sich, dieses Buch zu kaufen. Er wird an der Kasse jedoch überrascht, als er erfährt, dass das Buch teurer sein soll als die Preisangabe auf dem Etikett es aussagt. An der Kasse wurde der alte Preis berücksichtigt. Die Kassiererin ist verwirrt und der Kunde verärgert (siehe Abbildung 3.2).

Das Problem von redundanten Daten ist also, dass sie unterschiedlich geändert werden könnten, und dass man danach nicht einmal mehr weiß, welche eigentlich die aktuellen Daten sind.

In der Fachsprache bezeichnet man diese Problematik als die Gefahr von *Inkonsistenz*, d.h. Daten können sich widersprechen. Der Ressourcenverbrauch (Speicherkapazitäten) ist im Vergleich dazu ein eher sekundäres Problem.

Aus diesem Grund sollten Anstrengungen unternommen werden, um entsprechende Redundanz in einem IT-System zu vermeiden.

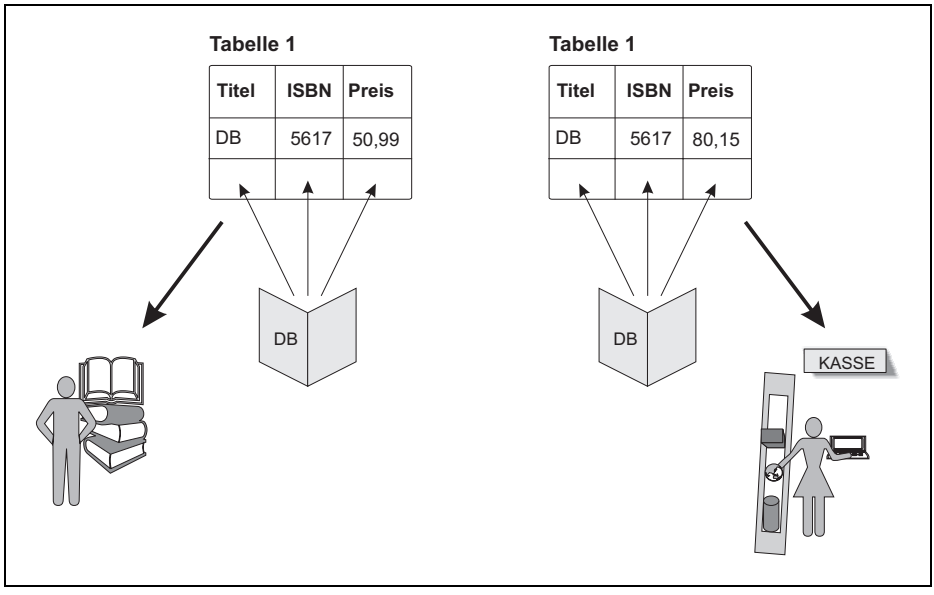


Abbildung 3.2: Fallbeispiel für Redundanz im Buchgeschäft

3.2 Konzepterstellung mit dem ER-Modell

Die Abbildung von Informationsstrukturen der realen Welt auf das konzeptuelle Schema einer Datenbank wird meist nicht direkt vorgenommen, sondern vielmehr über den Zwischenschritt der Abbildung auf ein semantisches Datenmodell, das dann auf das konzeptuelle Schema transformiert wird.

Das am häufigsten zu diesem Zweck eingesetzte Datenmodell ist das *Entity-Relationship Modell*, das in der heute gebräuchlichen Form 1976 von *P. Chen* vorgestellt wurde und in der Folge diverse Erweiterungen erfuhr. Das Entity-Relationship Modell geht grundsätzlich davon aus, dass sich die im Rahmen des Modells zu verwaltende Information als eine Menge von Objekten (Entities) beschreiben lässt, zwischen denen Beziehungen (Relationships) herrschen. Ähnlich wie das relationale Modell, sieht das Entity-Relationship Modell Attribute zur Beschreibung von Objekten und Beziehungen vor und macht Gebrauch vom *Primärschlüsselkonzept*. Darüber hinaus sind jedoch weitergehende Konzepte wie *Generalisierung* oder *Aggregation* vorhanden.

3.2.1 Die Elemente im ER-Modell

Das ER-Modell kommt mit einigen wenigen Elementen aus, die in diesem Abschnitt vorgestellt werden sollen.

Bezeichnung	Englische Bezeichnung	Bedeutung
Entitäten	entities	Eine Entität ist das Abbild eines Objektes aus der Realwelt.
Entitätstyp	entity type	Die Menge der Entitäten, die gleiche Eigenschaften haben und deshalb zusammengefasst werden können.
Beziehungen	relationships	Über die Beziehungen werden die Entitäten sinnvoll verknüpft.
Beziehungstyp	relationship type	Beziehungstypen stellen auch wieder eine Zusammenfassung der Beziehungen gleicher Eigenschaften dar.

Tabelle 3.1: Elemente des ER-Modells

Bei der Darstellung des ER-Modells in einem Diagramm werden die abgebildeten Symbole verwendet:

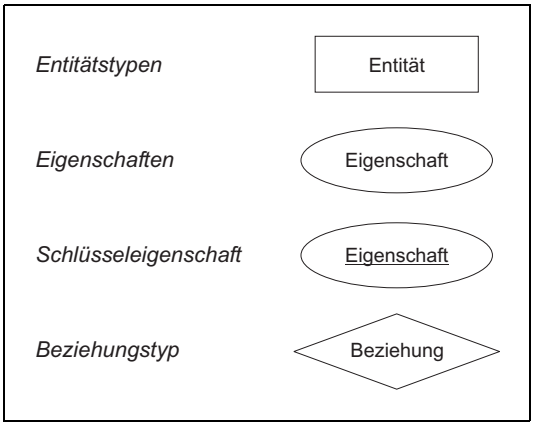


Abbildung 3.3: Symbole des ER-Modells

3.2.2 ER-Modell für die Beispieldatenbank

An dieser Stelle soll nun das Konzept für unsere Beispieldatenbank erstellt werden. Es handelt sich um die Strukturen eines Dienstleistungsunternehmen, welches die internen und externen Geschäftsabläufe automatisieren will. Die Datenbank soll in der Form konzipiert werden, dass sie auch für Firmen mit ähnlichen Geschäftsstrukturen verwendet werden kann.

Dem *Kunden* werden von den *Mitarbeitern* des Unternehmens verschiedene Dienstleistungen angeboten, die wir im Folgenden zusammenfassend *Service* nennen. Dieser einfache funktionale Zusammenhang wird nun in einem *ER-Diagramm* dargestellt.

1. Schritt: Selektion der Objekte

Abbildung 3.4: Selektion der Objekte

Zunächst werden die Entitäten (Objekte) herausgesucht und mit einem Rechteck als Symbol dargestellt. Mitarbeiter, Service und Kunde sind eindeutig identifizierbare Exemplare aus der Realwelt. Die Menge der Entitäten haben gleiche Eigenschaften und gelten dann als *Entitätstyp*. So hat ein Mitarbeiter aus der Buchhaltung, ebenso wie die Mitarbeiterin aus der Marketingabteilung, die Eigenschaften *MitarbeiterNr, Name* und *Geburtstag*. Dabei müssen die Werte dieser Eigenschaften nicht identisch sein. Im nächsten Schritt werden also den Objekten die Eigenschaften zugewiesen.

2. Schritt: Zuweisen der Eigenschaften

Üblicherweise sind für die verschiedenen *Entitäten* weitere Informationen, d.h. weitere Eigenschaften vorhanden.

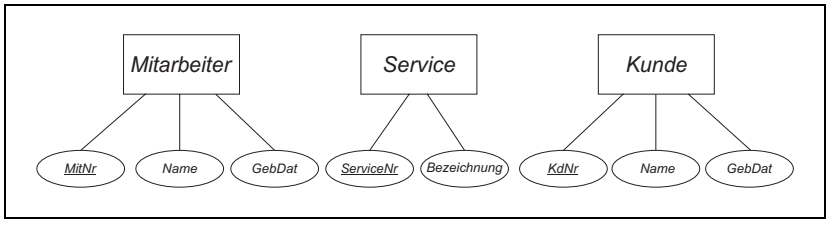


Abbildung 3.5: Zuweisen der Eigenschaften

Eine oder mehrere Eigenschaften eines Entitätstyps müssen als *Primärschlüssel* ausgewiesen werden. Keine Entität darf existieren, die nicht durch seinen Primärschlüsselwert eindeutig von allen anderen Entitäten seines Typs unterscheidbar ist. Bezeichner von Entitätstypen müssen untereinander eindeutig sein, Eigenschaftbezeichner müssen innerhalb eines Typen eindeutig sein. So darf innerhalb des Modells zwar die Eigenschaft *Name* sowohl für den Kunden als auch für den Mitarbeiter vergeben werden, jedoch ist es unzulässig, dem Mitarbeiter eine weitere Eigenschaft mit der Bezeichnung »Namen« zuzuweisen, die z.B. den Namen der Abteilung beinhaltet.

3. Schritt: Objekte mittels Beziehungstypen verknüpfen

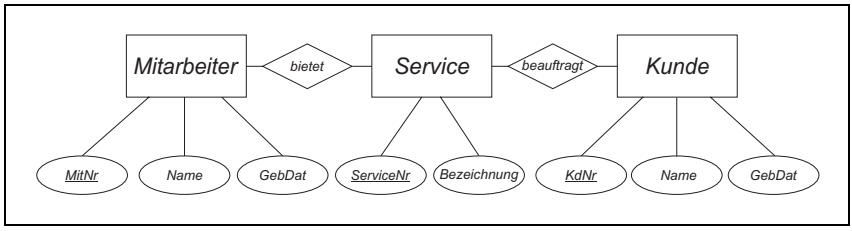


Abbildung 3.6: Objekte mittels Beziehungstypen verknüpfen

Beziehungen zwischen Entitäten werden durch sogenannte Beziehungstypen beschrieben, die ebenso wie Entitäten strukturierte Datenobjekte sind. Ein Beziehungstyp muss wenigstens zwei Entitätstypen zueinander in Beziehung setzen. Im Gegensatz zu den Entitätstypen ist ein Beziehungstyp jedoch kein eigenständiges Objekt, sondern kann nur existieren, wenn auch die referierten Entitätstypen existieren. In dem Beispiel werden die Entitäten *Service* und *Kunde* über einen Beziehungstypen mit der Bezeichnung »beauftragt« verknüpft. Die Bezeichnung des Beziehungstypen sollte so gewählt werden, dass sich daraus eine Aussage über die Art der Beziehung ableiten lässt. Dies erhöht die Lesbarkeit des Diagramms.

4. Schritt: Beziehungstypen kennzeichnen

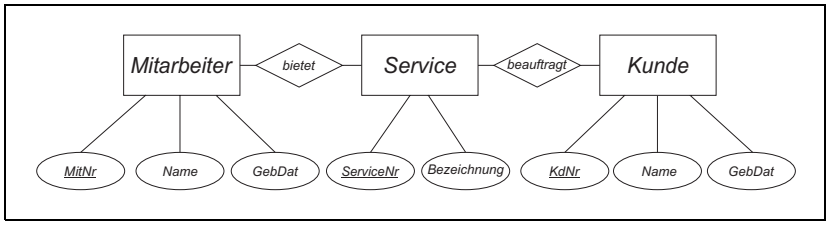


Abbildung 3.7: Beziehungstypen kennzeichnen

Schließlich werden im Diagramm die Beziehungstypen gekennzeichnet. In dem angegebenen Beispiel soll ein Mitarbeiter nur für jeweils eine Art von Serviceleistung zuständig sein. Umgekehrt kann aber ein und dieselbe Serviceleistung von mehreren Mitarbeitern angeboten werden. Daraus ergibt sich eine *1:n*-Beziehung, die entsprechend der Abbildung 3.7 im Diagramm gekennzeichnet wird.

3.2.3 Generalisierung

Dem Konzept der Bildung von *Untertypen* kommt durch den immer stärkeren Einfluss von objektorientierten Ansätzen bei der Datenbanktechnologie eine besondere Bedeutung zu.

Die Attributstruktur wird teilweise von anderen Typen geerbt, deren Elemente zugleich auch Elemente der vererbenden Typen sind. Im erweiterten Entity-Relationship Modell sind solche Zusammenhänge durch einen speziellen Relationship-Typen ausdrückbar, der als *Generalisierung* bezeichnet wird.

Eine Generalisierungsbeziehung besteht zwischen einem ausgezeichneten, im Folgenden als *Obertyp* bezeichneten Entitätstypen, und einer Menge von anderen Entitätstypen, die im Folgenden als Untertypen bezeichnet werden. Der Obertyp fasst dabei die Entitäten aller Untertypen zusammen. Er kann Eigenschaften besitzen, die dann den Untertypen gemein sind, und er kann wie jeder andere Entitätstyp bei der Bildung von Beziehungstypen verwandt werden.

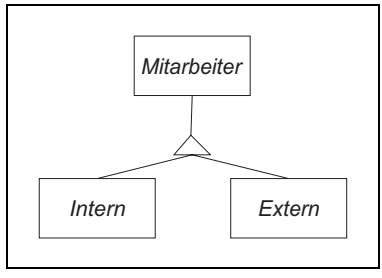


Abbildung 3.8: Beispiel der Generalisierung

Das Bestehen einer Generalisierungsbeziehung zwischen *Entitätstypen* wird durch ein Dreieck dargestellt, von dessen Spitze eine Kante zum Obertypen und von dessen Unterseite Kanten zu Untertypen ausgehen.

Das angegebene Beispiel stellt dar, dass Mitarbeiter eine Obermenge von internen und externen Mitarbeitern ist. »Interne« und »Externe« teilen gewisse Eigenschaften (z.B. Namen), weisen andererseits aber auch spezifische Eigenschaften auf, die sich aus der Art des Beschäftigungsverhältnisses ergeben.

Im konzeptuellen Entwurf spielen diese Verschärfungen meist aber eine geringe Rolle. Bedeutsam ist vielmehr, dass die Generalisierung eine Möglichkeit bietet, beim Entwurf von mehreren, speziellen Entitätstypen zugunsten eines Obertyps, der gemeinsame Merkmale bündelt, zu abstrahieren.

3.2.4 Aggregation

Da Datenobjekte oft aus anderen zusammengesetzt sind, ist die Darstellbarkeit von *aggregierten Objekten* eine wichtige Voraussetzung für die Modellierung von Datenstrukturen auf einem hohen, abstrakten Niveau. Während das ursprüngliche Entity-Relationship Modell keine speziellen Konzepte zur Darstellung von *Aggregationshierarchien* vorsah, kann eine geschachtelte Aggregation im erweiterten Entity-Relationship Modell durch Umwandlung von Beziehungstypen zu Entitätstypen bewirkt werden.

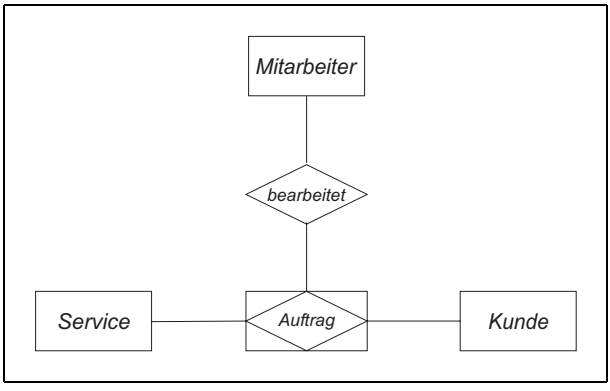


Abbildung 3.9: Beispiel der Aggregation

Ausgangspunkt der Erweiterung ist dabei, dass sich ein Beziehungstyp zwischen Entitäten als elementarer Konstruktor für aggregierte Objekte benutzen lässt, sofern man sie als ein aus den beteiligten Entitäten zusammengesetztes Datenobjekt auffasst. In diesem Sinne liegt es nahe, auch Beziehungstypen zuzulassen, die andere Beziehungstypen zueinander in Beziehung setzen. Da dies im Modell verboten ist, führt man im erweiterten Modell ein spezielles Symbol – die von einem Rechteck umschlossene Raute ein, die sowohl den Beziehungstyp als auch den Entitätstyp gleichermaßen symbolisiert.

In der Abbildung 3.9 wird ein Typ *Auftrag* dargestellt, der sich aus der Aggregation von *Kunden* und *Service* ergibt. Dieser neue Typ kann Eigenschaften besitzen, die sowohl dem Kunden als auch dem Service zuzuordnen wären. Ein typisches Beispiel für diesen Fall wäre das Datum an dem der Kunde einen Service beauftragt. Die neue Eigenschaft *Auftragsdatum* kann nun problemlos an den Typen *Auftrag* vergeben werden und bezieht sich auf Kunden und Service gleichermaßen.

3.2.5 Abbildung auf das relationale Datenmodell

Die Abbildung von ER-Modellen auf das relationale Datenmodell kann grundsätzlich vorgenommen werden, indem Entitätstypen und Beziehungstypen auf eigenständige Relationen (Tabellen) abgebildet werden.

Die Überführung eines Entitätstyps in eine eigenständige Relation ist bei vorgegebenen Attributen und Primärschlüssel leicht möglich. Das Modell aus Abbildung 3.7 würde umgesetzt in Tabellenform wie folgt aussehen:

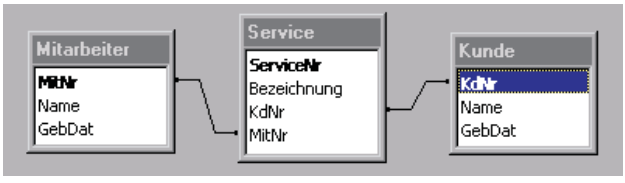


Abbildung 3.10: Die Tabellen Mitarbeiter, Service und Kunde

Die Spalten *MitNr*, *ServiceNr* und *KundenNr* bilden jeweils den *Primärschlüssel* in den drei Tabellen. Um eine Beziehung zwischen den Tabellen aufbauen zu können, müssen darüber hinaus Fremdschlüssel definiert werden. In der Tabelle *Mitarbeiter* ist die Spalte *ServiceNr* der Fremdschlüssel, der in Verbindung mit dem Primärschlüssel *ServiceNr* aus der Tabelle *Service* eine Verknüpfung zwischen den beiden Tabellen ermöglicht. Relativ leicht können so *1:1*- und *1:n*-Beziehungen erstellt werden. Eine *n:m*-Beziehung, wie sie zwischen den Objekten *Service* und *Kunde* vorliegt, ist, wie dies in Abbildung 3.10 geschehen ist, nicht realisierbar. Dazu müsste man mit einer Hilfstabelle arbeiten oder man bedient sich des Modells aus Abbildung 3.9, bei dem ein Beziehungstyp zum Entitätstyp aggregierte. Bei diesem Beispiel könnte die Tabellenstruktur ähnlich aussehen. Es wird eine weitere Tabelle *Auftrag* angelegt, so wie es in Abbildung 3.11 dargestellt ist.

Bei dieser Tabellenstruktur liegt zwischen den Tabellen *Kunde*, *Service*, *Mitarbeiter* und der neuen (Hilfs-) Tabelle jeweils eine *1:n*-Beziehung vor, wenn wir voraussetzen, dass jeweils nur ein Service beauftragt wird.

Bei der Überführung von Beziehungstypen (einschließlich Aggregationen) werden im Allgemeinen alle Eigenschaften des Beziehungstypen, sowie die Primärschlüssel der beteiligten Entitätstypen in eine eigenständige Relation überführt. Wenn Relationship-Typen eine *1:1*- oder *1:n*-Beziehung realisieren, kann aber auch darauf verzichtet werden und stattdessen der Relationship-Typ mit der Relation eines beteiligten Entity Typen adäquat verschmolzen werden. Der Primärschlüssel, der sich ergebenden Relation setzt sich dann auch aus den Primärschlüsseln der beteiligten Relationen zusammen und kann (falls dies das Datenbanksystem erlaubt) zusätzlich als Fremdschlüssel deklariert werden, um die referenzielle Integrität von Relationships zu modellieren.

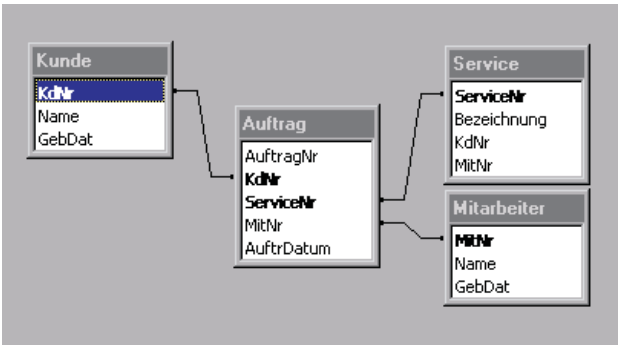


Abbildung 3.11: Abbildung der Entität Auftrag in eine Relation

Ebenfalls nicht direkt vorgesehen ist das Konzept der Generalisierung. Bei Generalisierungsbeziehungen werden daher im Allgemeinen nur Untertypen, die nicht wiederum Obertypen sind, in Relationen überführt. Diese Untertypen übernehmen die Attribute von ihren Obertypen.

Die Generalisierung lässt sich aber durch Projektion mehrerer Relationen auf gemeinsame Attribute, gefolgt von einer Vereinigung, operativ ausdrücken, was im Rahmen der Definition von externen Schemata durch Operatoren der relationalen Algebra zur Realisierung von »virtuellen« Generalisierungsrelationen ausgenutzt werden kann.

3.3 Optimieren des Datenbankentwurfs durch Anwendung der Normalformen

Vielfältige Untersuchungen zum relationalen Modell haben eine eigene Datenbanktheorie hervorgebracht, die sich unter anderem mit den Abhängigkeiten innerhalb der Tabellen beschäftigt. Diese Untersuchungen tragen dazu bei, Redundanzen und Anomalien aufzudecken oder helfen diese zu vermeiden.

Nach dem Entwurf einer relationalen Datenbank kann mit der Normalisierung Datenredundanz stufenweise reduziert werden. Beim Prozess der Normalisierung wird ein bereits bestehendes Schema mit Regeln, die aus der Relationenalgebra ableitbar sind, optimiert. Die Tabellenstruktur wird dermaßen angepasst, dass die eingangs erwähnten Qualitätsmerkmale auf das Gesamtschema zutreffen.

Jede Tabelle sollte so aufgebaut sein, dass das Hinzufügen oder Löschen einer Zeile (bzw. eines Eintrages in ihr) nicht Informationen verändert oder löscht, die anderwärts noch benötigt werden.

Treten solche Probleme auf, dann spricht man von *Modifikations-Anomalien*, d.h. es kommt zu unerwünschten Folgen, wenn Daten in einer Tabelle hinzugefügt, geändert

oder gelöscht werden. Solche Anomalien treten dann auf, wenn eine Tabelle Einträge enthält, die sinnvollerweise auf zwei oder mehr Tabellen verteilt werden sollten. Das Aufspalten einer Tabelle in zwei oder mehrere Tabellen, um solche Anomalien zu beseitigen, nennt man *Normalisierung* (Normalization).

Zwischen 1970 und 1981 betrieb man Forschung, wie man Modifikations-Anomalien vermeiden kann. Je nach Quelle der Modifikations-Anomalie, die man ausschließt, unterscheidet man seither verschiedene Normalformen:

- Erste Normalform (first normal form) (1NF),
- Zweite Normalform (second normal form) (2NF),
- Dritte Normalform (third normal form) (3NF).

Es gelten zunächst einige Grundbedingungen, die wie folgt zusammengefasst werden können:

- Tabellen sind zweidimensional mit Reihen und Spalten,
- jede Reihe enthält Daten, die zu einem Objekt oder einem Teil eines Objektes gehören,
- jede Spalte muss einen (in der Tabelle) einmaligen Namen tragen,
- keine zwei Reihen dürfen identisch sein,
- die Reihenfolge der Spalten und Reihen ist bedeutungslos.

3.3.1 Erste Normalform

Regel: *Eine Tabelle ist in der ersten Normalform, wenn sie den Grundbedingungen entspricht und die Einträge der Tabellenspalten einwertig (atomar) sind.*

Tabelle: Mitarbeiter

MitarbNr	Adresse
1001	Schmidt, Jochen, Wienerweg 3, Hamburg
1002	Meyer, Heidi, Amselstieg 12, Berlin
1003	Ehrhardt, Klaus, Kölnerweg 11, Berlin
⋮	⋮

Abbildung 3.12: Die Tabelle Mitarbeiter

Abbildung 3.12 zeigt uns eine Tabelle, die nicht dieser Regel entspricht. Die Spalte *Adresse* hat keine einwertigen Einträge. Mehrere Informationen, wie Name, Straße und Ort sind in einer Spalte zusammengefasst. Die nächste Abbildung zeigt uns, wie ein Entwurfsschema einer Tabellenstruktur in der ersten Normalform aussehen müsste.

Tabelle: Mitarbeiter

MitarbNr	AbtNr	Int_Ext	AdrNr
1001	1	I	221
1002	1	E	222
1003	2	I	223
⋮			

Tabelle: Adresse

AdrNr	Name	Vorname	Straße	Hnr	Ort
221	Schmidt	Jochen	Wiener Weg	3	Hamburg
222	Meyer	Heidi	Amselstieg	12	Berlin
223	Ehrhardt	Klaus	Kölnerweg	11	Berlin
⋮					

Abbildung 3.13: Tabellen Mitarbeiter und Adresse in der ersten Normalform

Die Adressinformationen sind in eine separaten Tabelle eingegangen. In beiden Tabellen gibt es nun ausschließlich Spalten, die einwertige Einträge vorweisen.

3.3.2 Zweite Normalform

Regel: *Eine Tabelle genügt der zweiten Normalform, wenn sie der ersten Normalform entspricht und Nicht-Schlüsselattribute voll funktional abhängig vom Schlüssel sind, und nicht nur von einer Teilmenge des Schlüssels.*

In der zweiten Normalform hängen alle Spalten (Attribute), die nicht zum Primärschlüssel gehören, vom Primärschlüssel als Ganzem ab. Alle Tabellen in der ersten Normalform, die einen einfachen (aus nur einer Spalte bestehenden) Primärschlüssel haben, sind so automatisch auch in der zweiten Normalform. Ist aber der Primärschlüssel zusammengesetzt, ist für die zweite Normalform zu prüfen, ob es Attribute gibt, die in Wirklichkeit nur von einer oder einem Teil der Spalten des Primärschlüssels abhängen. Ist dies der Fall, ist mit diesem Teil des Primärschlüssels eine neue Tabelle zu bilden und die ursprüngliche Tabelle in zwei Tabellen aufzuteilen.

Wir sehen uns die Abbildung 3.14 an. In der Tabelle *Mitarbeiter* bilden die Spalten *MitarbNr* und *AbtNr* einen zusammengesetzten primären Schlüssel. Die Spalte *Abteilung* ist nicht von beiden Schlüsselspalten, sondern nur von der *AbtNr* funktional abhängig. Damit ist sie nur von einer Teilmenge des Primärschlüssels abhängig und verstößt gegen die Regel der zweiten Normalform. Gibt es keinen zusammengesetzten Primärenschlüssel, dann ist es auch nicht erforderlich, die zweite Normalform anzuwenden. Im Beispiel aus Abbildung 3.14 wäre es jedoch erforderlich, erneut eine Aufteilung der Daten vorzunehmen. Zu diesem Zweck werden die Informationen über die Abteilung inklusive der Abteilungsnummer in einer neuen Tabelle *Abteilung* übertragen. Das

Ergebnis sehen wir in der Abbildung 3.15. Die Tabelle *Mitarbeiter* hat noch die Spalte *AbtNr*, die hier den Fremdschlüssel darstellt. Über diesen Fremdschlüssel kann eine Verknüpfung mit der Tabelle *Abteilung* erfolgen. Die Tabelle *Abteilung* kann ihrerseits weitere Attribute über den hier angegebenen *Namen* hinaus beinhalten.

Tabelle: Mitarbeiter

MitarbNr	AbtNr	Abteilung	Int_Ext
1001	1	Service	I
1002	1	Service	E
1003	2	Buchhaltung	I
⋮			

Abbildung 3.14: Tabelle Mitarbeiter

Tabelle: Abteilung **Tabelle: Mitarbeiter**

AbtNr	Name
1	Service
2	Buchhaltung
3	Marketing

MitarbNr	AbtNr	Int_Ext	AdrNr
1001	1	I	221
1002	1	E	222
1003	2	I	223
1004	3	E	224
1005	3	E	225

Abbildung 3.15: Die Tabellen Abteilung und Mitarbeiter in der zweiten Normalform

3.3.3 Dritte Normalform

Regel: *Eine Tabelle genügt der dritten Normalform, wenn sie der zweiten Normalform entspricht und jedes Nicht-Schlüsselattribut nicht-transitiv ist, d.h. es hängt (nur) direkt vom Primärschlüssel ab.*

Eine Tabelle in der dritten Normalform enthält keine *transitiven Abhängigkeiten*, d.h. Abhängigkeiten der Form, dass ein Attribut *A1* von einem anderen Attribut *A2* abhängt, welches wieder von einem anderen Attribut *A3* abhängt. Solche Abhängigkeiten können bewirken, dass das Löschen einer Reihe mit einem Eintrag von *A3* über *A2* zu einem Informationsverlust für *A1* führt.

Die Tabelle *Mitarbeiter* soll auch für diese Normalform als Beispiel dienen. Diesmal wird ein Teil der Adresse direkt dem Mitarbeiter zugewiesen. Die Adressnummer und die Ortsangabe sind Bestandteil dieser Tabelle. Weitere Angaben wie Straße, Hausnummer und PLZ sind aus Gründen der Übersichtlichkeit weggelassen. Die Spalte mit der Ortsangabe ist in diesem Fall nicht direkt vom Primärschlüssel (*MitarbNr*), sondern von der Spalte *AdrNr* abhängig. Es ist möglich, dass ein Mitarbeiter mehrere Wohnsitze hat. In der Tabellenstruktur aus Abbildung 3.16, wäre es nur auf Kosten von Redundanz möglich, dem Mitarbeiter mehrere Adressen zuzuordnen. So müsste für jeden Wohnsitz ein Datensatz in der Tabelle vorhanden sein, der dann jedesmal die Attribute *AbtNr* sowie *Int_Ext* mitschleppen würde, die nur einmal pro Mitarbeiter erforderlich sind.

Tabelle: Universal

MitarbNr	Name	Ort	Int_Ext	AbtNr	Abteilung
1001	Schmidt	Hamburg	I	1	Service
1002	Meyer	Berlin	E	1	Buchhaltung
1003	Ehrhardt	Berlin	I	2	Marketing
1003	Ehrhardt	Hamburg	I	2	Marketing

Abbildung 3.16: Universaltabelle

Die Lösung ist auch hier wieder eine Aufteilung in mehrere Tabellen. In Bild 3.13 sind bereits die Tabellen *Mitarbeiter* und *Adresse* verknüpft abgebildet. Diese Tabellenstruktur würde den Bedingungen der dritten Normalform genügen. Man spricht dann auch davon, dass die Tabelle(n) sich in der dritten Normalform befinden.

3.3.4 Weitergehende Normalformen

Über die bisher dargestellten Normalformen hinaus, die sich mit der Eliminierung funktionaler Abhängigkeiten befassen, existieren weitergehende Normalformen (vierte und fünfte Normalform) zum Ausschluss mehrwertiger Abhängigkeiten und sogenannter *Join Dependencies*, die aber in der Praxis kaum eine Rolle spielen. Sie werden deshalb an dieser Stelle nicht weiter behandelt. Selbst die ersten drei Normalformen können u.U. umgangen werden, wenn es aufgrund von Leistungskriterien sinnvoll ist. So können beispielsweise redundante Daten gezielt in eine Tabelle erzeugt werden, um über diese Zwischenwerte zu erhalten.

3.4 Physischer Entwurf

Bei einem physischen Entwurf geht es darum, das bereits erarbeitete logische Konzept an die gegebenen physischen Voraussetzungen anzupassen. Speicherstrukturen sowie Zugriffsmechanismen sind aufgrund der Datenbankarchitektur zu entwerfen. Wichtiger Faktor ist neben der geforderten Funktionalität der Versuch die Zugriffszeiten zu minimieren. Folgende Überlegungen sollten deshalb zu einem physischen Entwurf führen, der schließlich in der Testphase optimiert und in der Produktivphase »getunet« werden kann:

- Definition des internen Schemas,
- Verwendung der Dateiformate,
- Block- und Seitenzuweisung auf Platte,
- Indexstrukturen,
- Verteilung von Datenobjekten auf mehrere Platten (Clustering),
- Grundlagen des physischen Datenbankentwurfs,
- Fragmentierung in verteilten Datenbanken,
- Datenbank-Benchmarks,
- Festlegung von Speicherungsstrukturen und Zugriffsmechanismen,
- logische Speicherorganisation.

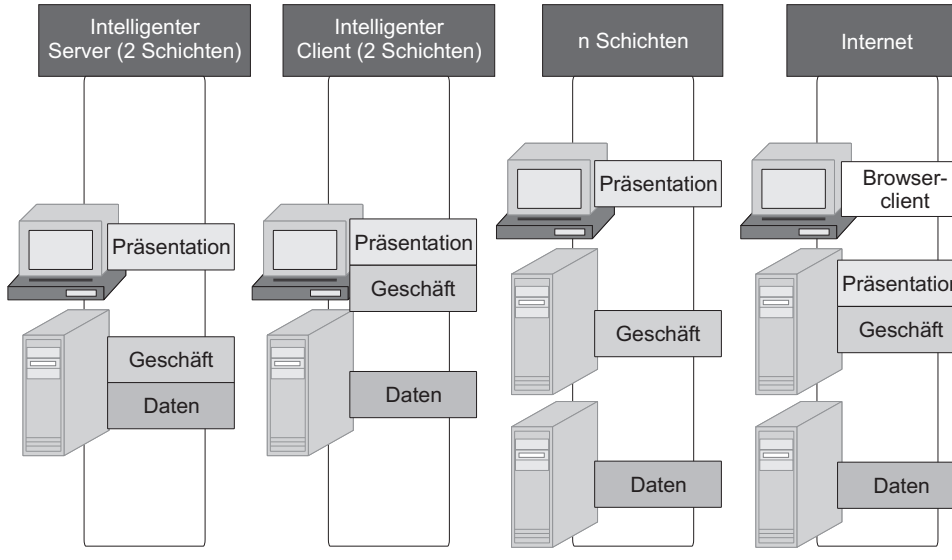


Abbildung 3.17: Systementwürfe für die Datenbankanwendung

Die Kenntnis der Geschäftsabläufe und die gegebenen Hardwarevoraussetzungen ergeben die nötigen Informationen für die Datenbankkonzepte. Die Abbildung 3.17 zeigt vier Entwürfe für eine Datenbankanwendung.

Beim »Intelligenten Server« trägt der Server die größte Last an Verarbeitungsprozessen. Die Geschäftslogik ist zum Großteil in der Datenbank implementiert. Haben die Client-Rechner nicht ausreichende Ressourcen, dann eignet sich dieser Entwurf für das System.

Bei der Lösung mit dem »Intelligenten Client« erfolgt ein Großteil der Verarbeitung auf den Client-Rechnern. Dadurch wird der Server entlastet, aber gleichzeitig steigt die Belastung des Netzwerkes. Transaktionen können so unter Umständen längere Ausführungszeiten haben.

Eine komplexere Lösung stellt der Entwurf mit n Schichten dar. Datenbank und Anwendungslogik sind hierbei auf getrennten Servern untergebracht. Bei komplexen Anwendungen wird somit die Last verteilt, was bei kleineren Anwendungen jedoch zu Leistungsverschlechterungen führen könnte.

Sollten die Informationen über einen Browser im Internet oder Intranet abgerufen werden, dann bietet sich die vierte Lösung mit einem Web-Server an.

3.5 Datenbankdiagramme bringen Struktur in den Entwurf

Komplexe Sachverhalte, die der Datenbankprogrammierer in der Praxis vorfindet, erfordern es, dass er die einzelnen Schritte seiner Arbeit dokumentiert. Besonders für Mitarbeiter, die nicht direkt an der Entwicklung beteiligt waren, ist es oftmals überhaupt nicht überschaubar, welche Tabelle nun welche Daten beinhaltet und welches Objekt aus der Realität abgebildet wird. Manchmal ist es einfacher, ein System neu zu implementieren, anstatt sich in ein schlecht dokumentiertes System einzuarbeiten. Einige wenige Diagramme sagen oftmals mehr als viele Worte. Zumindest ein Diagramm, welches die Abhängigkeiten der einzelnen Tabellen aufzeigt, sollte als Minimalbeschreibung vorhanden sein. Ein solches Diagramm liefert uns, wie bereits gesehen, der Entwurfprozess nach dem ER-Modell. Für objektorientierte Systeme sind das *Generische Semantische Modell* (GSM) oder das *Semantische Datenmodell* (SDM) Beschreibungsmöglichkeiten mit grafischer Unterstützung. Aber nicht nur für die Lesbarkeit nach der Entwicklung sind Diagramme erforderlich. Das Phasenmodell der Softwareentwicklung in Kapitel 2 verlangt nach der Entwurfsphase eine Spezifikation. Üblicherweise sind Diagramme Bestandteile der Spezifikation bzw. sie bilden die Grundlage für die Spezifikation. An dieser Stelle ist es also erforderlich, unseren konzeptionellen Entwurf, den wir bisher ansatzweise vollzogen haben, zu vervollständigen, um aus ihm eine Spezifikation für die Tabellenstruktur abzuleiten.

Das erste Teildiagramm für das Dienstleistungsunternehmen haben wir bereits in Abbildung 3.6 vorliegen. Hier wird die Beziehung zwischen Mitarbeiter, Service und Kunde beschrieben. In Abbildung 3.8 sind wir bereits einen Schritt weiter und haben durch die Aggregation aus der Beauftragung eine aggregierte Entität, nämlich den Auftrag erhalten. Ein weiteres Teildiagramm zeigt die folgende Abbildung.

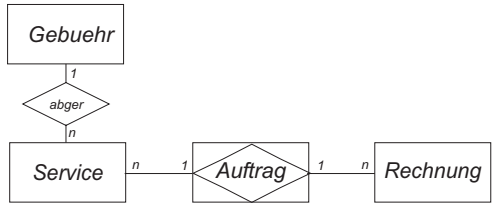


Abbildung 3.18: Darstellung der Abhängigkeiten von Service, Rechnung und Auftrag

Jeder beauftragte Service wird in Rechnung gestellt. Dabei wird jeder Service nach eine Gebührenordnung abgerechnet. Jede Rechnung wird genau einem Auftrag zugewiesen. Andrerseits können für jeden Auftrag mehrere Rechnungen erstellt werden. Daher haben wir zwischen Auftrag und Rechnung eine *1:n*-Beziehung. Die Dienstleistungen (Service) haben eine eindeutig abzurechnende Gebühr. Diese Gebühr kann allerdings auch als Abrechnungsgrundlage für andere Leistungen herangezogen werden. Daraus ergibt sich eine *1:n*-Beziehung auch für die Entitäten *Gebühr* und *Service*. Die *Leistungen* sollen jeweils genau einem *Auftrag* unterstellt sein. Auch hieraus ergibt sich eine *1:n*-Beziehung bei Service und Auftrag. Die Attribute sind in diesem Teildiagramm, wie auch in den anderen, nicht berücksichtigt.

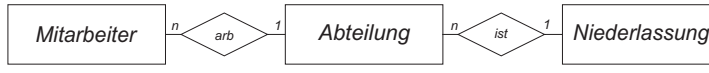


Abbildung 3.19: Abhängigkeiten zwischen Mitarbeiter, Abteilung und Niederlassung

Die Abbildung 3.19 bringt eine Übersicht der Abhängigkeiten zwischen *Mitarbeiter*, *Abteilung* und *Niederlassung*. Jeder Mitarbeiter arbeitet in ausschließlich einer Abteilung und diese wiederum wird eindeutig einer Niederlassung zugeordnet.

In dem Teildiagramm in Bild 3.20 wird dem Kunden eine Adresse und Bankverbindung zugewiesen. Die Adresse wird erweitert durch Angaben der Telekommunikation in der Entität *Telekom* sowie Angaben zum Land durch die Entität *Land*.

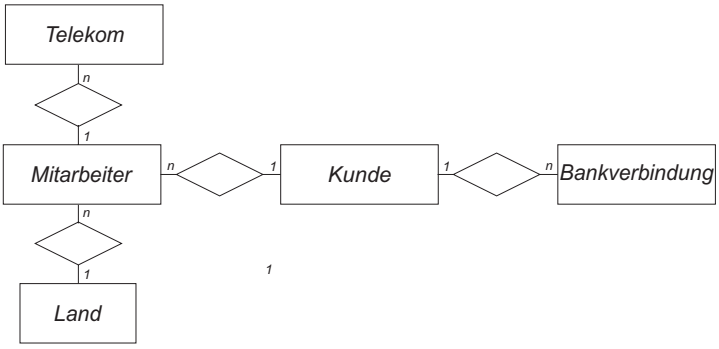


Abbildung 3.20: Teildiagramm mit Bankverbindung

Bei strukturell aufwändigen Entwürfen ist es üblich, das DB-Diagramm in mehrere Teildiagramme aufzuteilen. Für eine Gesamtübersicht ist es dann ratsam, auf Details zu verzichten. So kann in einem Gesamtdiagramm, wie in Abbildung 3.21, auf Attribute und Relationships verzichtet werden.

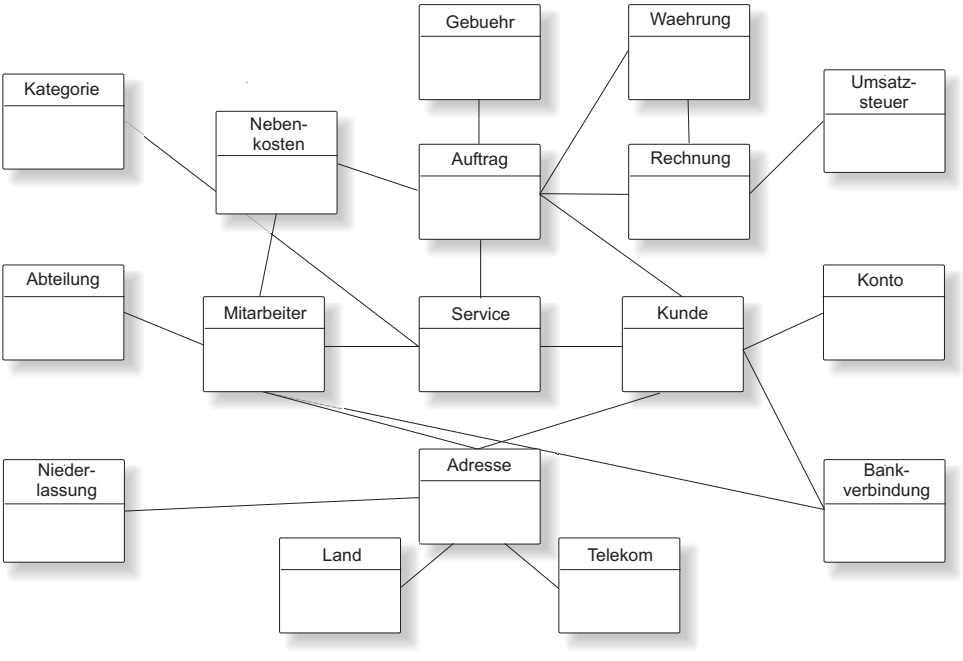


Abbildung 3.21: Gesamtübersicht für das Dienstleistungsunternehmen

In der Abbildung 3.21 sind die Teildiagramme zusammengefasst und um einige untergeordnete Entitäten erweitert. Die Entität *Konto* beispielsweise beinhaltet das Gutha-

ben bzw. den zu zahlenden Betrag des Kunden an das Unternehmen. Dies hat nichts mit dem Konto bei der entsprechenden Bank zu tun. Die Angaben zum Bankkonto gehören zur Entität *Bankverbindung*. Für die Erstellung der Rechnung ist die Information über Umsatzsteuer und auch Währung notwendig, deshalb sind hierfür zwei weitere Entitäten vorgesehen. Diese Angaben sind auch abhängig von den Vereinbarungen im Auftrag (deshalb besteht auch eine Abhängigkeit zwischen Auftrag und Währung).

Jeder Service gehört zu einer Kategorie, die sich auch in der Gebührenverordnung widerspiegelt, aus einem vom Unternehmen angebotenen Katalog an Leistungen. Also muss auch dieser Umstand mit einer weiteren Entität berücksichtigt werden. Dem Unternehmen entstehen nicht nur Kosten, die sich direkt aus der Arbeitszeit für die beauftragte Leistung ergeben, sondern Reisekosten und Materialkosten müssen auch an den Kunden weitergereicht werden, deshalb wird die Entität *Nebenkosten* ebenso in das Konzept aufgenommen.

3.6 Werkzeuge für den Entwurf

Für den Entwurf oder auch für das Erstellen von Diagrammen und Grafiken gibt es von verschiedenen Anbietern Werkzeuge, welche die Arbeit erleichtern.

Auf der Begleit-CD finden Sie das Programm ERM. Dies ist ein Werkzeug, mit dem Sie Ihre ER-Modelle erstellen können.

3.6.1 MS Access

Datenbankdiagramme können auch in MS Access nach dem Entwurf erstellt werden. Wenn Sie auf EXTRAS/BEZIEHUNGEN... klicken, dann wird das Fenster BEZIEHUNGEN geöffnet (siehe Abbildung 3.22).

Dort haben Sie durch Klicken auf die Schaltfläche TABELLE ANZEIGEN, die Möglichkeit, die zuvor entworfenen Tabellen im Diagramm einzufügen. Wählen Sie eine oder auch mehrere Tabellen aus und klicken Sie dann auf die Befehlsschaltfläche HINZUFÜGEN, um die Tabellen einzufügen. (siehe Abbildung 3.23)

Schließlich kommen Sie durch Positionieren der einzelnen Tabellen und Erzeugung der Beziehungen zu einem Ergebnis, welches in Abbildung 3.23 als Teilstruktur genügen soll.

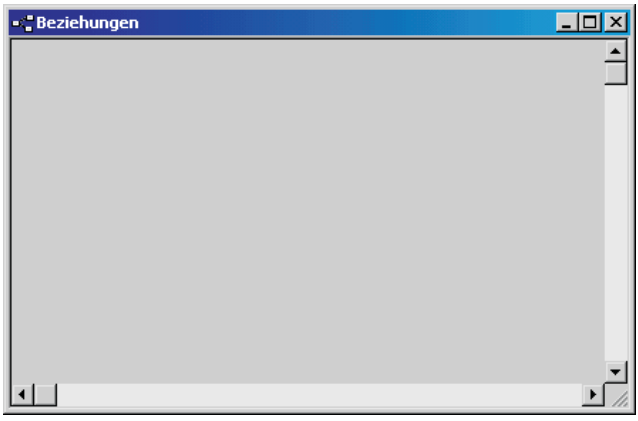


Abbildung 3.22: Das Fenster »Beziehungen« in MS Access

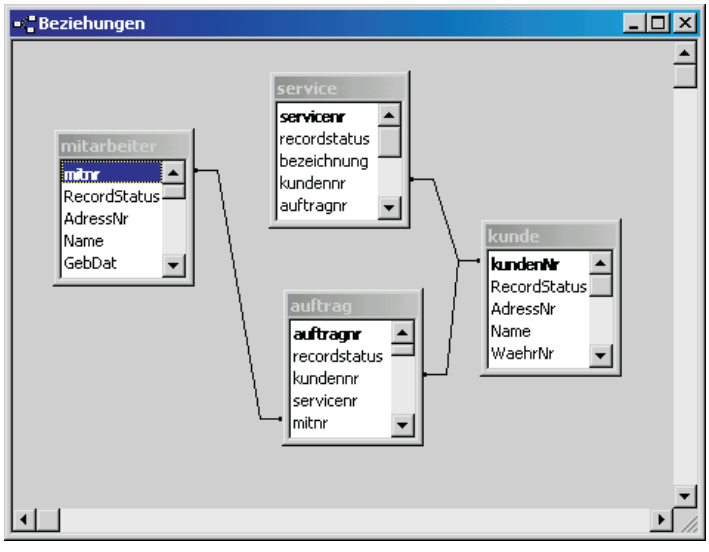


Abbildung 3.23: Beziehungen zwischen Access-Tabellen

3.6.2 MS Visio

Microsoft Visio ist ein nützliches Werkzeug für die Diagrammerstellung. Mit Visio können Sie die vielfältigen, täglich anfallenden Aufgaben klar umreißen und dokumentieren, anderen Ihre Ideen mitteilen und Informationen gemeinsam nutzen. Mit Visio stellen Sie diese Informationen in einer Form dar, die leicht zu erfassen ist und auch im Gedächtnis bleibt. In Visio Standard erstellte Diagramme ermöglichen wertvolle Einblicke in bestehende Prozesse, Projekte und Informationen zu Mitarbeitern

und helfen einzelnen Mitarbeitern sowie Teams auf diese Weise bei einer effizienteren Zusammenarbeit. Und wenn Sie Visio-Diagramme in Ihre Office-Dokumente integrieren, werden Ihre Mitteilungen noch prägnanter, die wichtigsten Punkte können noch besser hervorgehoben werden, und kulturelle Unterschiede und technische Hindernisse können leichter überwunden werden. Abbildung 3.24 zeigt zwei Diagramme, die mit Visio erstellt wurden.

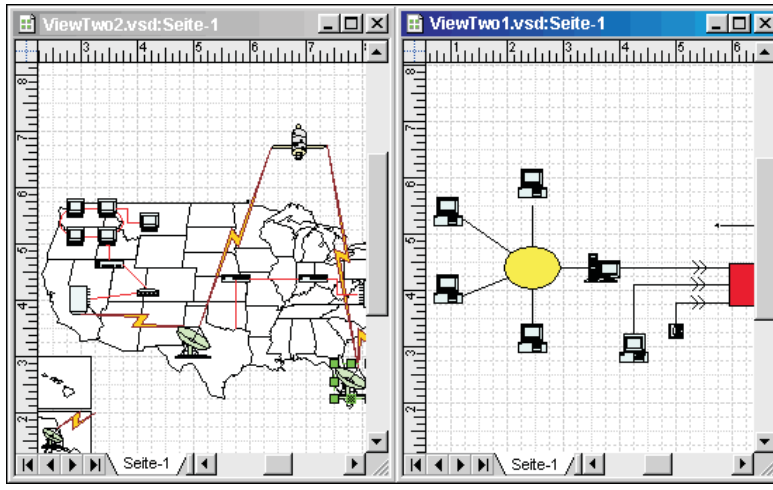


Abbildung 3.24: Diagramme in Visio

3.7 Vom Entwurf zur Spezifikation

Nach Fertigstellung des Entwurfes werden die Ergebnisse in die Erstellung der Spezifikation eingebracht. Die Spezifikation wird auch häufig als Feinentwurf bezeichnet. Komponenten für Rahmenwerke, die Berücksichtigung von Standards und die Unabhängigkeit gegenüber proprietären Umgebungen erfordern eine genaue Spezifikation der zu erstellenden Software. Aus dieser abstrakten Spezifikation wird eine Implementierung für die jeweilige Umgebung (Sprache, Rahmenwerke, etc.) erstellt. Dieser Schritt gehört zu den fehlerträchtigsten Tätigkeiten beim Erstellen von Software. Das liegt daran, dass bei der Umsetzung des Entwurfs nicht nur das Wissen um die zu verwendende Umgebung, sondern auch das Fachwissen aus dem Problembereich vorhanden sein muss. Sollen dabei sogar Konzepte wie z.B. Entwurfsmuster oder Optimierungen eingesetzt werden, steigen die Anforderungen an den Implementierungsschritt nochmals erheblich.

Es wurden Techniken evaluiert und ein System entworfen, das eine automatische Umsetzung abstrakter Spezifikationen ermöglicht. Mit Hilfe eines Graphenregelsys-

tems (GRS) wird die Software-Spezifikation, die als UML-Diagramm (Unified Modeling Language) formuliert werden kann, auf die Implementierungsebene abgebildet. Dazu wird eine Regelbasis verwendet, die die Abbildung bestimmter Konzepte beschreibt. Diese Regelbasis kann ebenfalls als UML-Graph beschrieben werden, was die Anpassung und Erweiterung der Regeln vereinfacht. Ein sich daraus ergebendes Problem ist die Synchronisation der abstrakten Spezifikation und der dazugehörigen Implementierung. Die Erfahrung zeigt, dass Änderungen an der Software nicht nur auf der Ebene der Spezifikation, sondern auch direkt in der Implementierung erfolgen. Um sicherzustellen, dass die Spezifikation immer zur aktuellen Implementierung passt, wird derzeit ein Verfahren entwickelt, das Änderungen an der Implementierung auf der abstrakten Ebene mitführt.

3.8 Zusammenfassung

Bei der Entwicklung einer Datenbank wird das Ziel angestrebt, einen Ausschnitt der realen Welt abzubilden. In diesem kleinen Weltausschnitt existieren eine Vielzahl von Objekten, wie Personen, Unternehmungen, Produkte oder Dienstleistungen. Diese stehen in unterschiedlichen Beziehungen zueinander.

Im ersten Schritt werden die Objekte der realen Welt, die für die Aufgabenstellung relevant sind, mit ihren Beziehungen untereinander in abstrakter Weise beschrieben, d.h. modelliert. Um diesen Vorgang visuell zu unterstützen, werden Datenmodelle mit Hilfe des ER-Modells erstellt.

Damit die abstrakten Objekte und Beziehungen noch deutlicher werden, wird dieser Modellansatz durch ein ER-Diagramm unterstützt.

Im zweiten Schritt wird aus der Datenmodellierung das logische Datenbankmodell erstellt. Man spricht hier auch von der sogenannten konzeptionellen Phase. Auf welches Datenbankmodell man dabei abzielt, liegt daran, welches Datenbankverwaltungssystem später zum Einsatz kommen soll.

Bei der Konzeptionierung des logischen Datenbankmodells gibt es schon eine Ausrichtung auf das Datenbankverwaltungssystem, das man später verwenden will.

Im dritten Schritt wird das logische Datenbankmodell dann in der Datenbeschreibungssprache des Zielsystems umgesetzt. Hierbei werden Tabellen angelegt, wobei durch die Felddefinition die Datenbeschreibung erfolgt.

Nach dem Entwurf einer relationalen Datenbank kann man mit der Normalisierung Datenredundanz stufenweise reduzieren. Beim Prozess der Normalisierung wird ein bereits bestehendes Schema durch Anwendung einiger Regeln optimiert.

Alle Normalformen sind von großer theoretischer Bedeutung. In der Praxis beschränkt man sich jedoch auf die ersten drei.

Interessant wird die Kombination aus semantischem Datenmodell und logischem Datenbankmodell, wenn bei der Modellierung mit dem Entity-Relationship Modell eine saubere Definition eines Entitäten-Beziehungsmodells erfolgt. Werden hierbei nämlich die Abbildungsregeln beachtet, dann sind die Normalformen jederzeit automatisch erfüllt.

Dieser Modellierungsansatz gewährleistet uns redundanzfreie, gegen Anomalien gesicherte Datenbankmodelle. Die Implementierung, d.h. die Realisierung solcher Modelle lässt sich so leichter in eine konkrete Datenbank umsetzen.

4 Datenbanksprache SQL

Das Konzept für die Datenbank ist entworfen. Jetzt muss dieses Konzept abgebildet werden auf eine Datenbank. Die Datenbank selbst, sowie die Tabellen und anderen Datenbankobjekte müssen erzeugt werden. Bei einigen DBMS ist dies mittels grafischen Werkzeugen möglich, so dass hier keine tiefgreifenden Kenntnisse der Datenbanksprache erforderlich sind. Oftmals ist es jedoch erforderlich, die Tabellen, Indizes, Schlüssel etc. explizit durch Anweisungen zu erstellen. Sämtliche Aufgaben, die an einer relationalen Datenbank erledigt werden, können über eine Datenbanksprache abgewickelt werden. SQL (*Structured Query Language*) ist eine standardisierte Abfragesprache, die alle erforderlichen Sprachelemente enthält, um sämtliche Arbeiten, die beim Umgang mit einer relationalen Datenbank anfallen, auszuführen. Erste Versionen von SQL hießen SEQUEL, SEQUEL2 und waren aus SQUARE hervorgegangen, einer eher mathematisch orientierten Sprache. Da sich SQL im Laufe seiner Geschichte weiterentwickelt hat, kann man jedoch nicht von einem einzigen Standard sprechen.

Die Mächtigkeit von SQL erhöht sich mit der neuen Generation (SQL-99) bedeutend; neue Optionen gibt es in mehreren Bereichen der Basisdefinition.

Die Einführung in den Sprachgebrauch von SQL wird sich in diesem Kapitel zunächst auf SQL-92 beziehen. Dies ist zum einen darin begründet, dass die in diesem Buch vorgestellten DBMS diesen Standard unterstützen und zum anderen sind die grundlegenden Definitionen aus SQL-92 auch in SQL-99 erhalten geblieben und behalten somit ihre Gültigkeit. Am Ende des Kapitels wird auf die Erweiterungen von SQL-99 eingegangen.

Es gibt verschiedene herstellerspezifische Abwandlungen vom Standard SQL-92. TSQL ist z.B. ein SQL-Dialekt, der aus dem Hause Microsoft stammt und beim MS SQL Server verwendet wird. PL/SQL dagegen ist das Pendant von Oracle. Die grundsätzliche Struktur der Sprache ist aber bei beiden erhalten geblieben. In Kapitel 4.6 wird auf markante Unterschiede bei den Dialekten hingewiesen, die dem Datenbankprogrammierer, in Hinblick auf Plattformunabhängigkeit, das Leben erschweren.

4.1 Standard SQL

Die in den weiteren Ausführungen gemachten Aussagen zu SQL beziehen sich auf den momentan gültigen Standard SQL-92. SQL ist relational vollständig, d.h. der Sprachumfang ist äquivalent zu dem der Relationenalgebra. Die verschiedenen Befehle können vereinfacht in drei Kernbereiche unterteilt werden:

- Data Manipulation Language (DML),
- Data Definition Language (DDL),
- Data Control Language (DCL).

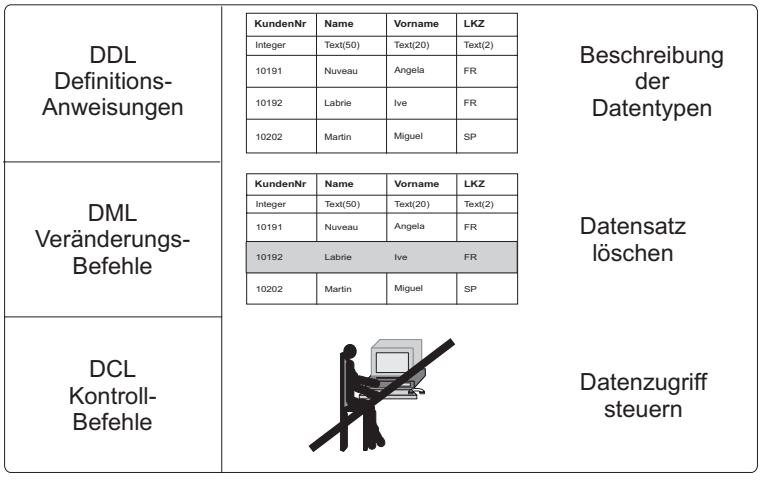


Abbildung 4.1: Die drei Kernbereiche von SQL

Wenn wir uns die Datenbankarchitektur nach ANSI/SPARC von Abbildung 2.12 nochmals vor Augen führen, dann können wir den einzelnen Ebenen die zugehörigen SQL-Befehle zuordnen. Mit *DDL-* und *DML-Anweisungen* werden die *externe* und die *konzeptuelle Ebenen* angesprochen. Die Kontrollbefehle aus dem *DCL-Bereich* steuern die interne Ebene. Um bei den vielen neuen Begriffen nicht die Übersicht zu verlieren, wollen wir zu jedem Bereich ein Beispiel behandeln (siehe auch Abbildung 4.1).

Beispiel 4.1:

Beim Datenbankentwurf haben wir die Entität Kunde mit ihren Attributen und Beziehungen erstellt. Diese wird nun in die Tabellenstruktur gebracht. Grundsätzliche Überlegungen haben ergeben, dass die einzelnen Attribute immer die gleiche Domäne haben, d.h. in der Spalte *Name* haben wir immer Daten, die dem Typ *Text* oder *Character* entsprechen. Innerhalb einer Tabelle soll nun jeder Spalte ein Datentyp zugewiesen werden. Dies wird mit SQL-Anweisungen aus dem *DDL-Bereich* realisiert.

Beispiel 4.2:

Ein Mitarbeiter ist aus der Firma ausgeschieden. Seine Daten sollen deshalb aus der Datenbank entfernt werden. In der Tabelle *Mitarbeiter* ist also ein Datensatz zu löschen. Daten werden bei diesem Beispiel verändert, darum handelt es sich um eine *DML-Anweisung*. Die SQL-Anweisung, die dies ermöglicht, werden wir später noch besser kennen lernen.

Beispiel 4.3:

Kundendaten sollen nicht für jeden Mitarbeiter und schon gar nicht im Internet abrufbar sein. Diese Daten sollen besonders geschützt werden und nur einem gewissen Personenkreis zugänglich gemacht werden. Auch dies ist mit den jeweiligen SQL-Anweisungen machbar. Wir nennen diese Art von Befehlen deshalb *Kontrollbefehle (DCL)*.

4.2 Allgemeine Sprachelemente

Neben den SQL-Anweisungen, die wir den drei benannten Kernbereichen zuordnen können, gibt es weitere Sprachelemente. Dazu gehören *Datentypen*, *Variablen*, *Operatoren*, *Funktionen*, *Sprachkonstrukte zur Ablaufsteuerung* und *Kommentare*.

4.2.1 Datentypen

Wenn man mit Daten in einer DB und insbesondere mit Daten aus anderen Quellen arbeiten will, muss vorher so etwas wie eine Normierung stattgefunden haben. Man spricht hier auch von einer *Typisierung der Vorgänge* und damit letztendlich auch der ihnen zugrundeliegenden Daten.

Man kann also sagen, dass der Datentyp die Eigenschaften des Datenobjektes festlegt, und auch die Anzahl und Art von Operationen die mit einem Datentyp durchgeführt werden können. Der Datentyp `Integer` stellt z.B. die Menge der ganzen, positiven und negativen natürlichen Zahlen dar. Man spricht hier auch vom *Wertebereich* eines Datentyps. Die Größe der maximal darstellbaren Zahl hängt allerdings vom verwendeten System ab.

Bevor nun diese Daten für eine Operation herangezogen werden, muss man sich überlegen, ob das Endergebnis oder auch Teilergebnisse zu *Typverletzungen* führen könnten. Ein simples Beispiel soll diese Notwendigkeit verdeutlichen.

Beispiel 4.4:

Der Vorname und Nachname des Kunden soll in einer Operation mit Zeichenketten als einfaches Attribut zusammengefasst werden. Das neue Attribut hat die Bezeichnung Namen und soll eine Zeichenlänge von 20 haben. Wenn also, der Vorname schon

15 Zeichen umfasst, bleiben für den Nachnamen logischerweise nur noch 5 Zeichen, was mit Sicherheit zu Abschneidungen beim Nachnamen führen würde. Grundsätzlich muss man sich also überlegen, ob die vorgegebene Zeichenlänge für das Attribut Name nicht vergrößert werden kann oder andernfalls die Verknüpfungsoperation nicht besser so aufgebaut wird, dass die Abschneidung, wenn nötig, beim Vornamen auftritt.

Die Anforderungen an unsere heutigen Systeme führt zu immer neuen Datentypen, die passend zu ihrer jeweiligen beabsichtigten Verwendungen sind. Probleme beim Festlegen der Datentypen liegen nicht nur in der *Redundanz*, auch die Gefahr der *Dateninkonsistenz* und eine *Ineffizienz* in der Datenverarbeitung und -verwaltung sind möglich. Differenzen in der Datenstruktur führen häufig dazu, dass man bei der Portierung auf eine andere Plattform eine Schnittstelle für die Anwendung programmieren muss.

Jeder Spalte einer Tabelle kann ein Datentyp zugewiesen werden. So wird schon aufgrund der Struktur verhindert, dass Daten nicht korrekt im System verarbeitet werden (siehe Domänenintegrität). SQL unterstützt die folgenden skalaren Datentypen:

Datentyp	**Beschreibung**
CHARACTER (n)	Zeichenkette fester Länge mit genau n Zeichen (n>0).
CHARACTER VARYING (n)	Zeichenkette variabler Länge mit bis zu n Zeichen (n>0).
BIT (n)	Bitkette fester Länge mit genau n Bits (n>0).
BIT VARYING (n)	Bitkette variabler Länge mit bis zu n Bits (n>0).
NUMERIC (p, q)	Dezimalzahl mit Vorzeichen und m Ziffern sowie angenommenem Dezimalpunkt q Ziffern von rechts (0<q<p, p>0).
DECIMAL (p, q)	Dezimalzahl mit Vorzeichen und m Ziffern sowie angenommenem Dezimalpunkt q Ziffern von rechts (0<q<p<m, p>0, siehe Anmerkung unten zur Erläuterung von m).
INTEGER	Vorzeichenbehaftete ganze Dezimal- oder Binärzahl (siehe unten).
SMALLINT	Vorzeichenbehaftete ganze Dezimal- oder Binärzahl (siehe unten).
FLOAT (p)	Gleitkommazahl N, etwa repräsentiert durch einen vorzeichenbehafteten Binärteil f an m Binärstellen (-1<f<+1, 0<p<m, siehe Anmerkung unten zur Erläuterung von m) und einem vorzeichenbehafteten binären ganzzahligen Exponenten e, etwa N= f* (10 ** e). Hinweis: Das Symbol »**« steht für Exponentiation. Man beachte aber, dass SQL faktisch keinen solchen Operator unterstützt (siehe Abschnitt 7.4).

Tabelle 4.1: Datentypen in SQL-92

Je nach Datenbank werden weitere Datentypen unterstützt. Hierzu gibt es weitere Informationen unter »Datenbankdialekte« oder »multimediale Datentypen«.

4.2.2 Variablen

Variablen sind Sprachelemente mit zugewiesenen Werten. Lokale Variablen werden insbesondere in Stapeln oder Prozeduren verwendet. Dort werden sie vom Benutzer definiert und über die SET-Anweisung wird ihr ein Wert zugewiesen. Lokale Variablen verlieren ihre Gültigkeit außerhalb der Prozedur oder des Stapels.

Beispiel 4.5:

```
SET @var = 555
```

In Beispiel 4.5 wird der lokalen Variablen `@var` der Wert 555 zugewiesen.

4.2.3 Operatoren

Daten werden erfasst, um sie zu verarbeiten. Sollen die Stimmen gezählt werden, die ein Kandidat bei einer Wahl bekommen hat, so muss wenigstens die Möglichkeit der Addition vorhanden sein. Es werden also Operationen durchgeführt, bei denen existierende Daten verglichen, umgeformt oder auch zusammengefasst werden. In SQL gibt es festgelegte Zeichenfolgen für diese Operationen, die man dann auch Operator nennt. Je nach Art der Operationen gibt es verschiedene Kategorien für die Operatoren:

Arithmetische Operatoren

Die arithmetischen Operatoren in SQL-92 sind in Tabelle 4.2 zu sehen:

Operator	**Beschreibung**
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo Operation für den Rest einer ganzzahligen Division

Tabelle 4.2: Arithmetische Operatoren in SQL-92

Vergleichsoperatoren

Mit den Vergleichsoperatoren werden Werte verglichen.

Operator	Beschreibung
=	Test auf Gleichheit
!=,^=,<>	Test auf Ungleichheit
<,>,<=,>=	Test auf kleiner/größer (gleich) als
IN	Gleichheit zu einem Mitglied oder einer Menge
NOT IN	invertiert den Wert den Operators IN.
ANY	vergleicht den Wert mit jedem Wert aus einer Liste (vorausgehen muss eine der Operatoren: =,!=,<,>,<=,>=). Mindestens ein Vergleich muss den Wert TRUE liefern.
ALL	vergleicht einen Wert mit allen Werten aus einer Liste (vorausgehen muss eine der Operatoren: =,!=,<,>,<=,>=). Alle Vergleiche in der Ergebnismenge müssen den Wert TRUE liefern.
[NOT] BETWEEN x AND y	[nicht] größer oder gleich x und kleiner als oder gleich y.
[NOT] EXISTS	liefert TRUE, wenn eine Unterabfrage mindestens eine [keine] Zeile zurückgibt.
[NOT] LIKE	liefert TRUE, wenn der Operand einem Muster entspricht.
IS [NOT] NULL	Test auf NULL-Wert.

Tabelle 4.3: Vergleichsoperatoren in SQL-92

Logische Operatoren

Logische Operationen testen den Wahrheitswert einer Bedingung und geben einen booleschen Wert (richtig oder falsch) zurück.

Operator	Rückgabewert = Wahr, wenn
AND	beide booleschen Ausdrücke TRUE ergeben.
NOT	invertiert das Ergebnis eines logischen Ausdrucks.
OR	mindestens einer der beiden booleschen Ausdrücke TRUE ergibt.

Tabelle 4.4: Logische Operatoren in SQL-92

Sonstige Operatoren

Operator	Beschreibung
UNION	kombiniert zwei Unterabfragen.
INTERSECT	Mengendurchschnitt der selektierten Zeilen.
MINUS	Mengendifferenz zwischen zwei selektierten Mengen.
DISTINCT	eliminiert doppelte Zeilen oder doppelte Werte in einem Aggregatausdruck.

Tabelle 4.5: Sonstige Operatoren in SQL-92

4.2.4 Funktionen

SQL bietet eine Reihe von Funktionen, die bei der Verarbeitung der Daten zur Verfügung stehen. Auch diese lassen sich wieder aufteilen in mehrere Kategorien. Einige der Funktionen dürften aus der Mathematik bzw. anderen Programmiersprachen bereits bekannt sein.

Aggregatfunktionen

Diese Funktionen beruhen (mit Ausnahme der Funktion COUNT(*)) auf einem Aggregat, was hier so viel heißt wie Sammlung skalarer Werte aus einer Spalte. Als Ergebnis erhält man einen einzelnen skalaren Wert.

Funktion	Beschreibung
COUNT	Anzahl skalarer Werte in der Spalte
SUM	Summe der skalaren Wert in der Spalte
AVG	Durchschnitt der skalaren Werte in der Spalte
MAX	Größter skalarer Wert in der Spalte
MIN	Kleinster skalarer Wert in der Spalte

Tabelle 4.6: Aggregatfunktionen in SQL-92

Numerische Funktionen

Funktion	Beschreibung
ABS(n)	Absolutwert
MOD(m,n)	Modulo-Funktion
EXP(n)	Exponentialfunktion
LN(n)	Natürlicher Logarithmus
LOG(m,n)	Logarithmus von n zur Basis m
COS(n)	Cosinus von n
SIN(n)	Sinus von n
TAN(n)	Tangens von n
SQRT(n)	Quadratwurzel von n
SIGN(n)	Signum-Funktion

Tabelle 4.7: Numerische Funktionen in SQL-92

Zeichenkettenfunktionen

Dieser Funktionstyp wird zur Veränderung der Datenwerte innerhalb einer Spalte verwendet. Die Syntax der Funktionen ist sehr stark von der des DB-Herstellers abhängig.

Funktion	Beschreibung
CHR(n)	konvertiert das entsprechende Zeichen aus dem ASCII-Code.
LOWER(s)	konvertiert eine Zeichenkette vollständig in Kleinbuchstaben.
UPPER	konvertiert eine Zeichenkette vollständig in Großbuchstaben.
SUBSTR(s,m[,n])	liefert den Teilstring des String s ab der Position m mit der Länge n. Beim Weglassen von n wird die Länge von m bis zum rechten Ende des Strings verwendet.

Tabelle 4.8: Zeichenkettenfunktionen in SQL-92

4.2.5 Ablaufsteuerung

Der logische Ablauf innerhalb eines SQL-Skriptes kann mit einigen SQL-Schlüsselanweisungen gesteuert werden. So kann beispielsweise mit `BEGIN...END`-Blöcken eine Reihe von SQL-Anweisungen zusammengehalten und als Einheit behandelt werden. Es wird unterschieden zwischen Sprachkonstrukten auf der Anweisungsebene, wie `BEGIN...END`-Blöcke, `IF...ELSE`-Blöcke oder dem `WHILE`-Konstrukt und denen auf der Zeilenebene, wie es die `CASE`-Anweisung ist.

4.2.6 Kommentare

Besonders bei SQL-Skripten, die nicht sonderlich verständlich sind, ist es ratsam mit Kommentaren zu arbeiten. Kommentare können innerhalb einer SQL-Anweisung eingefügt werden, dann werden sie *Inline-Kommentare* genannt oder man fügt einen *Blockkommentar* ein, der sich über mehrere Zeilen erstreckt und bei dem der Anfang und das Ende markiert werden. Als Beispiel sind sowohl Inline- als auch Blockkommentare für den MS SQL Server angeführt.

Beispiel 4.6:

```
/*
** Dies ist ein Beispiel eines Blockkommentares beim SQL
** Server
** Die folgende SQL-Anweisung ermittelt die
** Niederlassungsnummer der Niederlassung,
** welche den Auftrag mit der Nr.: 1256 bearbeitet
*/
SELECT NiederlassungNr FROM Niederlassung
WHERE AuftrNrAktuell = 1256
```

Beispiel 4.7:

```
SELECT NiederlassungNr
, Beispiel mit einem Inlinekommentar
FROM Niederlassung WHERE AuftrNrAktuell = 1256
```

So sollte das gesamte Skript an geeigneten Stellen mit Kommentaren versehen werden. Diese sind so zu platzieren, dass auch andere Personen die von den Anweisungen durchgeführten Aktionen verstehen können.

4.3 DDL-Datendefinition

Die SQL-Anweisungen aus dem Bereich der Datendefinition dienen der Einrichtung, Modifikation oder Entfernung von Datenbankobjekten. Es werden bei der Einrichtung nur »leere« Strukturen geschaffen, d.h. die Objekte sind noch nicht mit Daten gefüllt. So werden bei einem Tabellenobjekt unter anderem Spaltenname und der Wertebereich für die Spalten definiert, es werden jedoch nach einer DDL-Anweisung keine Daten in der Tabelle stehen, wenn diese nicht mittels einer DML-Anweisung eingefügt wurden.

Bei der Neuererstellung einer Datenbank wird also zunächst die Struktur, die zuvor konzeptionell beschrieben wurde, durch DDL-Befehle aufgebaut.

In SQL gibt es drei Typen von Anweisungen, mit denen man Datenbankobjekte strukturieren kann:

- `CREATE`: Erzeugung von Objekten,
- `ALTER`: Nachträgliche Änderung,
- `DROP`: Löschen der Objekte aus dem Datenbankschema.

Die aufgeführten Schlüsselwörter werden an den Beginn der SQL-Anweisung gestellt. Es folgt der Typ und der Name des Datenbankobjektes. Schließlich werden weitere Parameter zur Datendefinition angegeben. In den folgenden Abschnitten ist die Syntax vereinfacht dargestellt und Beispiele für verschiedene DBMS angegeben.

4.3.1 CREATE TABLE

Mit dieser Anweisung wird eine leere matrixförmige Tabelle erzeugt. Die gewählte Struktur wird im Systembereich des DBMS gespeichert, d.h. die Art der Struktur und die Daten selbst sind physikalisch voneinander getrennt.

Syntax:

```
CREATE TABLE table (column type [DEFAULT value]
     [CONSTRAINT column_constraint] [, ... ] )
     [CONSTRAINT table_constraint]
     [ON COMMIT
     {DELETE | PRESERVE}
      ROWS ]
```

Beschreibung:

CREATE TABLE
Schlüsselbezeichnung für diese SQL-Anweisung.

Table
Name der Tabelle, die erzeugt werden soll. Der Name muss den Richtlinien für die Bezeichnung im jeweiligen DBMS genügen.

Column
Spalte oder durch Komma getrennte Liste mit Spalten, welche die Tabelle beinhalten soll.

Type
Für die Spalte zulässiger Datentyp.

DEFAULT value
Default-Wert (wird kein Default-Wert angegeben ist die Nullmarke, der Default).

CONSTRAINT
Schlüsselwort für die Einschränkungs-Klausel.

Column_constraint
Bezeichnung für die Einschränkung (muss datenbankweit eindeutig sein).

Table_constraint
Bezeichnung für die Einschränkung auf Tabellenebene.

ON COMMIT...ROWS
Aktion bei einer Transaktion, die durch DELETE oder PRESERVE gekennzeichnet wird.

DELETE
Ist DELETE an dieser Stelle gesetzt, wird eine temporäre Tabelle bei jedem COMMIT automatisch geleert.

PRESERVE
Ist PRESERVE an dieser Stelle gesetzt, wird eine temporäre Tabelle bei jedem COMMIT unverändert gelassen.

Für die Beispieldatenbank würde eine SQL-Anweisung für die Erstellung einer Tabelle mit Telekommunikationsdaten wie im angegeben Beispiel aussehen.

Beispiel 4.8:

```
CREATE  TABLE  telekom
(telekomnr int  identity (1,1) not null CONSTRAINT pk_telekomnr primary key,
     recordstatus int,
     adressnr int,
     kategorienr int,
     kategorie char(20),
     kurzname char(20),
     nummer char(50),
     createuserid char(10),
     modifyuserid char(10),
     createdate datetime,
     modifydate datetime);
```

Erzeugt wird hier die Tabelle *telekom* mit den Feldern und den entsprechenden Datentypen für die einzelnen Felder. Dem ersten Feld *telekomnr* wird neben dem Datentyp auch eine Einschränkung und ein primärer Schlüssel zugewiesen. Die Einschränkung besagt in diesem Fall, dass in dieser Spalte, also *telekomnr*, kein NULL-Wert vorkommen darf. Eine weitere Besonderheit für diese Spalte ist die Vorgabe eines Autowertes. Dies geschieht hier (TSQL) mit der *identity*-Anweisung. Für jeden neuen Datensatz in dieser Tabelle wird automatisch ein neuer Wert für *telekomnr* vergeben, beginnend mit dem Wert »1« und jeweils um die Schrittweite »1« inkrementiert.

Dieses Beispiellisting können sie direkt im Query Analyzer vom MS SQL Server ausprobieren. Wer mit PostgreSQL arbeitet, muss ein paar Anpassungen vornehmen, die grundsätzliche Syntax bleibt jedoch erhalten.

Nach Ausführung der SQL-Anweisung liegt die Tabelle *telekom* – wie in Abbildung 4.2 gezeigt – vor. Wie bereits erwähnt, wurden noch keine Werte erzeugt, deshalb ist die Tabelle selbstverständlich leer.

telekomnr	recordstatus	adressnr	kategorienr	kategorie	kurzname	nummer	create...

Abbildung 4.2: Ergebnis der CREATE TABLE-Anweisung

4.3.2 CREATE INDEX

Indizes werden hauptsächlich aus zwei Gründen erstellt:

- Bessere Performance beim Zugriff auf häufig verwendete Spalten,
- Sicherstellung der Eindeutigkeit von Spalten.

Im später folgenden Kapitel »Indizes« wird diese Problematik nochmals aufgegriffen. Mit der Anweisung `CREATE INDEX` wird sekundäre Speicherstruktur, der Index, erzeugt.

Syntax:

```
CREATE [ UNIQUE ] INDEX index ON table
  ( column [ ASC | DESC ]
    [ , column [ ASC | DESC ] ... ]
    [ STORAGE Storageclause ]
   )
```

Beschreibung:

CREATE INDEX

Einleitende Schlüsselbezeichnung für diese SQL-Anweisung.

UNIQUE

Ist UNIQUE gesetzt, überprüft das System, ob Duplikate vorhanden sind (falls Daten bereits in der Tabelle vorhanden sind) oder es wird bei jedem Hinzufügen eines neuen Datensatzes auf Duplikate überprüft. Sind Duplikate vorhanden, so wird eine Fehlermeldung ausgegeben und die Aktion wird nicht durchgeführt.

index
Bezeichnung für den erstellten Index (nach gültigen Bezeichnungsregeln).

table
Name der Tabelle für die ein Index erzeugt werden soll.

column
Spalte oder Kommaliste mit mehreren Spalten, auf die ein Index gesetzt wird.

ASC
Sortierreihenfolge ist aufsteigend.

DESC
Sortierreihenfolge ist absteigend.

STORAGE Storageclause
Angabe zur Speicherstruktur.

Beispiel 4.9:

```
CREATE INDEX idx_nummer
     ON telekom (nummer);
```

Auf die Spalte *nummer* der Tabelle *telekom* wird erwartungsgemäß häufiger eine Abfrage angesetzt, deshalb soll diese Spalte auch mit einem Index versehen werden. Im Beispiel 4.9 wird dies durch die CREATE INDEX-Anweisung erledigt.

4.3.3 CREATE VIEW

Eine *Sicht* ist eine »virtuelle« Tabelle, die es ermöglicht, Daten aus einer oder mehreren Tabellen oder Sichten zusammenzufassen und anzuzeigen oder zu verändern. Diese Sichten haben keine physische Existenz und Änderungen wirken sich nur auf die Basistabellen aus. Mit `CREATE VIEW` können Sichten definiert werden.

Syntax:

```
CREATE VIEW view [ column [, ...] ]
   AS SELECT bedingungsausdruck [ AS colname ] [, ...]
   FROM table [ WHERE condition ]
   [ WITH [ CASCADE | LOCAL ] CHECK OPTION ]
```

Beschreibung:

CREATE VIEW
Schlüsselbezeichnung für diese SQL-Anweisung.

view
Bezeichnung der Sicht (nach gültigen Richtlinien den DBMS).

column
Spalte oder Liste von Spaltennamen, die Element dieser Sicht sein sollen.

AS SELECT
Klausel, welche die Auswahlabfrage einleitet.

bedingungsausdruck
Ausdruck, der den Übergabewert liefert. Hier können auch das »*«-Zeichen, Wertangaben, oder Aggregatfunktionen stehen.

AS colname
Vergabe eines Aliasnamens.

FROM table
Tabelle, aus der die Übergabewerte stammen.

WHERE condition
Bedingung für die Abfrage.

WITH ... CHECK OPTION
Alle INSERT- und UPDATE-Aktionen werden auf Integrität überprüft.

LOCAL
Überprüfung auf Integrität erfolgt nur auf die angegebene Sicht.

CASCADE
Überprüfung auf Integrität erfolgt auch auf von der Sicht abhängige Sichten.

Beispiel 4.10:

```
CREATE VIEW qryTelekom
  AS SELECT * FROM telekom
  WHERE kategorie = 'Telefon';
```

Eine Sicht wird genauso wie eine Tabelle aufgerufen. Die Anweisung, die die oben gezeigte Sicht aufrufen würde, finden wir beim nächsten Beispiel.

Beispiel 4.11:

```
SELECT * FROM qryTelekom;
```

adressnr	kategorienr	kategorie	kurzname	nummer
1	3	Telefon	Meyer	0221-56346
2	3	Telefon	Wühlbeck	0511-452310
5	3	Telefon	Schmidt	06321-98765
12	3	Telefon	Hinz	05321-23411

Abbildung 4.3: Ergebnismenge der SELECT-Anweisung

Wie das Ergebnis zeigt, werden zunächst alle Spalten der Tabelle *telekom* beim Aufruf der Sicht angezeigt (in Bild 4.3 sind nicht alle Spalten dargestellt). Man kann die Auswahl jedoch weiter einschränken, indem man die nachgestellte SELECT-Anweisung modifiziert. So ist es sicherlich sinnvoller, wenn der Benutzer nur eine Sicht auf die für ihn relevanten Daten bekommt. Dazu ein weiteres Beispiel:

Beispiel 4.12:

```
CREATE VIEW qrytelekom
  AS SELECT kategorie, kurzname, nummer FROM telekom
  WHERE kategorie = 'Telefon';
```

Mit der SQL-Anweisung, die wir weiter oben im Beispiel 4.11 zur Anzeige der Sicht benutzt hatten, wollen wir auch hier das Ergebnis anzeigen lassen.

kategorie	kurzname	nummer
Telefon	Meyer	0221-56346
Telefon	Wühlbeck	0511-452310
Telefon	Schmidt	06321-98765
Telefon	Hinz	05321-23411

Abbildung 4.4: Ergebnismenge der Abfrage mit WHERE-Klausel

4.3.4 ALTER TABLE

Auch bei großer Sorgfalt, die für das Konzeptmodell aufgebracht wurde, ist es oft notwendig, die Struktur einer Tabelle nachträglich zu verändern. Ebenso könnte zusätzlicher Speicherplatz mit der Anweisung `ALTER TABLE` reserviert werden.

Syntax:

```
 ALTER TABLE table
[ADD
     ( Column Element | Constraint Clause )]
[MODIFY
     (Column [Element] | [Constraint Clause]]
[DROP Drop Clause]
```

Beschreibung:

ALTER TABLE
Schlüsselklausel, mit der die SQL-Anweisung eingeleitet wird.

table
Tabelle, bei der eine Strukturänderung vorgenommen wird.

ADD
Schlüsselwort, welches das Hinzufügen von Elementen kennzeichnet.

MODIFY
Schlüsselwort, welches die Veränderung von Elementen kennzeichnet.

DROP
Schlüsselwort, welches das Entfernen von Elementen kennzeichnet.

Column Element
Spalte, die zugefügt wird oder an der eine Änderung vorgenommen wird.

Constraint Clause
Kennzeichnung der Einschränkung, die dem Objekt zugefügt wird.

In TSQL würde eine entsprechende Anweisung folgendermaßen aussehen:

Beispiel 4.13:

```
ALTER TABLE telekom
ADD
CONSTRAINT fk_adressnr
FOREIGN KEY (adressnr)
REFERENCES adresse(adressnr);
```

An dieser Stelle gibt es für diejenigen, die in SQL nicht geübt sind, etwas Klärungsbedarf. Zunächst ist es wichtig, dass die Tabellen *telekom* und *adresse* leer sind. Ansonsten

kann es zu Fehlermeldungen kommen. Datensätze in der Tabelle *telekom* müssen referenzierte Datensätze in der Tabelle *adresse* haben, um eine Schlüsseldefinition, wie in diesem Beispiel, vorzunehmen. Der Tabelle wird nachträglich ein *Fremdschlüssel* zugewiesen, der den Namen *fk_adressnr* hat und nach dem Schlüsselwort `CONSTRAINT` definiert wird. Nach dem Schlüsselwörtern `FOREIGN KEY` wird die Spalte, in diesem Fall *adressnr*, angegeben, die mit dem Schlüssel versehen wird. Schließlich werden im letzten Teil der Anweisung Tabelle und Name der Spalte angegeben, die den *primären Schlüssel* beinhaltet. Es wurde also eine Referenz zwischen den Tabellen *telekom* und *adresse* jeweils über die Spalten *adressnr* aufgebaut.

Die ALTER-Anweisung lässt weitere Variationen bzw. weitere Veränderungen an Datenbankobjekten zu. So kann auch nachträglich die Anzahl der Spalten einer *Sicht* verändert werden. Wie ein solcher Befehl aussehen würde, zeigt uns das folgende Beispiel.

Beispiel 4.14:

```
ALTER VIEW qry_telekom
  AS
  SELECT telekomnr, kategorie, kurzname, nummer FROM telekom
  WHERE kategorie = 'Telefon';
```

Beim Verändern der Sicht muss die gesamte SELECT-Anweisung nach dem Schlüsselwort `AS` erscheinen. Auch hier soll das Ergebnis wieder mittels einer Abfrage angezeigt werden.

telekomnr	kategorie	kurzname	nummer
7	Telefon	Meyer	0221-56346
8	Telefon	Wühlbeck	0511-452310
9	Telefon	Schmidt	06321-98765
10	Telefon	Hinz	05321-23411

Abbildung 4.5: Datensätze aus der Tabelle telekom

Zu den bereits definierten Spalten *kategorie*, *kurzname* und *nummer* kommt, durch die Neudefinition, die Spalte *telekomnr* hinzu.

In den beiden Beispielen zum ALTER-Befehl wurde bisher etwas hinzugefügt. Gleichwohl ist es möglich und auch aus der oben angeführten Syntax ablesbar, dass Teildefinitionen wieder gelöscht werden können. Dies gilt für Spalten in einer Tabelle oder einer Sicht und auch für Einschränkungen. Die Schlüsseldefinition, die in der Tabelle *telekom* für die Spalte *adressnr* vorgenommen wurde, soll im nächsten Beispiel durch eine ALTER-Anweisung wieder rückgängig gemacht werden.

Beispiel 4.15:

```
ALTER TABLE telekom DROP CONSTRAINT fk_adressnr;
```

4.3.5 DROP TABLE

Soll jedoch nicht nur eine Spalte oder ein Schlüssel der Tabelle, sondern die Tabelle als Ganzes gelöscht werden, muss man sich der DROP TABLE-Anweisung bedienen. Diese Anweisung dient dem Löschen einer Tabelle inklusive ihrem Inhalt.

Synatx:

```
DROP TABLE table [CASCADE CONSTRAINTS]
```

Beschreibung:

DROP TABLE
Schlüsselbegriffe der Anweisung.

table
Name der zu löschenden Tabelle.

CASCADE constraints
Die abhängigen Sichten und Einschränkungen werden ebenfalls gelöscht.

Die Handhabung ist vergleichsweise einfach, deshalb soll hier ein Beispiel genügen. Bevor eine Tabelle gelöscht wird, sollte man sich über alle Abhängigkeiten der Tabelle informieren.

Beispiel 4.16:

```
DROP TABLE telekom;
```

Jede Sicht auf die gelöschte Basistabelle muss explizit mit DROP gelöscht werden.

4.3.6 DROP VIEW

Wenn eine Sicht nicht mehr benötigt wird, kann ihre Definition mit der DROP VIEW-Anweisung aus der Datenbank entfernt werden. Beim Löschen einer Sicht werden ihre Definition und alle, ihr zugewiesenen Berechtigungen entfernt. Wird vom Benutzer eine Sicht abgefragt, die auf eine gelöschte Sicht verweist oder selbst gelöscht wurde, so kommt es zu einer Fehlermeldung.

Synatx:

```
DROP VIEW view { RESTRICT | CASCADE }
```

Beschreibung:

DROP VIEW
Schlüsselworte der Anweisung.

view
existierende Sicht, die gelöscht werden soll.

RESTRICT
Überprüfung auf Abhängigkeiten. Bei bestehenden Abhängigkeiten wird nicht gelöscht.

CASCADE
Abhängige Sichten oder Einschränkungen werden ebenfalls gelöscht.

Beispiel 4.17:

```
DROP VIEW qryTelekom;
```

Wird die SQL-Anweisung aus Beispiel 4.17 in einem RDBMS ausgeführt, so wird die Sicht *qryTelekom* inklusive der Berechtigungen gelöscht.

4.4 DML Manipulation der Daten

Diese Gruppe von Anweisungen bewirkt das Verändern von Daten. Dazu gehört neben der INSERT-, UPDATE- und DELETE-Anweisung auch die SELECT-Anweisung. Deshalb ist die Bezeichnung *Manipulation Language* etwas irreführend, denn hier werden die Daten nicht im eigentlichen Sinn manipuliert.

4.4.1 Die SELECT-Anweisung

Wie man dem Schlüsselwort *SELECT* bereits unschwer entnehmen kann, handelt es sich bei dieser Anweisung um eine Selektion (Auswahl) von Daten. Aus einer oder mehreren Tabellen oder Sichten können mit Hilfe von Operationen wie Verbund, Durchschnitt, Differenz, Selektion und Projektion, Ergebnismengen erzeugt werden. Die SELECT-Anweisung wird in den verschiedensten Variationen sehr häufig verwendet und ist auch im Sub-Bereich anderer Anweisungen zu finden. Um der ganzen Komplexität dieser Anweisung gerecht zu werden, wird es in diesem Kapitel noch einen weiteren Abschnitt mit erweiterten Abfragemöglichkeiten geben. Hier nun ein kurzer Einstieg.

Syntax:

```
SELECT [ ALL | DISTINCT ]
 { * | table.* | bedingungsausdruck }
 [ , { * | table.* | bedingungsausdruck } ] ...
FROM table [ , table ] ...
 [ WHERE condition ]
 [ GROUP BY bedingungsausdruck [ HAVING condition ] ]
 [ ORDER BY { bedingungsausdruck |position }
 [ ASC |DESC ]   ... ]
```

Beispiel 4.18:

```
SELECT * FROM adresse;
```

Diese simple Abfrage liefert eine Ergebnismenge, die in der folgenden Abbildung zu sehen ist. Sämtliche Daten, die in der Tabelle *adresse* stecken werden zur Anzeige gebracht.

AdressNr	RecordStatus	Typ	Name	Kurzname	PersonArt	StrassePostfach	Land	Plz	Ort	AnschriftAnrede
4	1	Kunde	Heiner	Heiner	2	Meisenweg 3	D	40032	Essen	Herr
5	1	Kunde	Klaus	Loops	2	Wacholderweg 23	D	34550	Herborn	Herr

Abbildung 4.6: Ergebnismenge der SELECT-Abfrage

Die Grundform einer Abfrage-Operation lautet:

`SELECT` <Spalte> *was?*
`FROM` <Tabelle> *woher?*
`WHERE` <Bedingung> *welche Bedingung?*

Im Beispiel weiter oben wurden die Fragen nach dem »was?« und »woher?« beantwortet. Diese gehören auch zu den Minimalangaben der SELECT-Anweisung, d.h. Spalten und Tabelle müssen immer angeben werden. Für sämtliche Spalten einer Tabelle haben wir den Platzhalter »*« kennen gelernt. Anstatt diesem, können alle Spalten aufgelistet werden, was jedoch einem vernunftbegabten Menschen schnell als überflüssige Schreibarbeit deutlich wird. Die zusätzliche WHERE-Bedingung ist optional. Die Abfrage kann durch eine Bedingung weiter eingeschränkt werden, sie muss es aber nicht. Die ohnehin recht kärgliche Ergebnismenge aus Abbildung 4.7 soll weiter eingeschränkt werden. Alle Adressen (in diesem Fall nur eine) aus dem Ort Essen sollen ausgegeben werden. Der Befehl mit der zusätzlichen WHERE-Bedingung würde dann folgendermaßen aussehen:

Beispiel 4.19:

```
SELECT * FROM adresse WHERE Ort ='Essen';
```

Im Folgenden werden die Hauptkomponenten des Befehls beschrieben:

SELECT
Schlüsselwort für die Auswahlabfragen.

ALL
gibt an, dass das Resultat doppelte Zeilen enthalten darf. ALL ist die Standardeinstellung.

DISTINCT
gibt an, dass im Resultset nur eindeutige Zeilen vorkommen dürfen.

FROM
Die FROM-Klausel legt Tabellen, Sichten, abgeleitete Tabellen und verknüpfte Tabellen für die SELECT-Anweisung fest.

WHERE
legt eine Suchbedingung fest, die die zurückgegebenen Zeilen einschränkt.

GROUP BY
Die GROUP BY-Klausel dient der Zusammenfassung von Datensätzen für die Auswertung durch statistische Funktionen.

HAVING
legt eine Suchbedingung für eine Gruppe oder ein Aggregat fest. `HAVING` wird in der Regel zusammen mit der GROUP BY-Klausel verwendet. Wenn `GROUP BY` nicht verwendet wird, verhält sich `HAVING` wie eine WHERE-Klausel.

ORDER BY
legt die Sortierung für das Resultset fest.

ASC
legt fest, dass die Werte in der angegebenen Spalte in aufsteigender Reihenfolge vom niedrigsten bis zum höchsten Wert sortiert werden sollen.

DESC
legt fest, dass die Werte in der angegebenen Spalte in absteigender Reihenfolge, vom höchsten bis zum niedrigsten Wert, sortiert werden sollen.

In einem weiteren Beispiel soll die GROUP BY-Klausel verständlich gemacht werden:

Beispiel 4.20:

```
SELECT kundennr, SUM(anzstunden)
FROM auftrag
GROUP BY kundennr;
```

Die Gesamtzahl der Arbeitsstunden für die jeweiligen Kunden soll ermittelt werden. In der Tabelle *auftrag* finden wird die entsprechenden Daten. Die Anzahl der Stunden

wird mit der Aggregatfunktion SUM berechnet. Dabei werden alle Datensätze berücksichtigt, die die gleiche *kundennr* haben. Gruppiert wird schließlich nach der *kundennr*, so wie wir es in der GROUP BY –Klausel des Beispiels sehen. Die Ergebnismenge sehen wir in Abbildung 4.7.

	kundennr	(No column name)
1	BEKL0001	414
2	HHHE0001	275

Abbildung 4.7: Ergebnismenge der gruppierten SELECT-Abfrage

Als etwas störend wird der Betrachter die Überschrift der zweiten Spalte empfinden. Spalten, welche das Ergebnis einer Aggregatfunktion enthalten, müssen einen Namen explizit zugewiesen bekommen. Damit diese Spalte eine sinnvolle Bezeichnung bekommt, muss folgende Anweisung ausgeführt werden.

Beispiel 4.21:

```
SELECT kundennr, SUM(anzstunden) AS Stundensumme
FROM auftrag
GROUP BY kundennr;
```

Im Vergleich zum vorherigen Beispiel wurde nur der Zusatz `AS Stundensumme` eingefügt. Damit wurde, wie wir in Bild 4.8 sehen, der erkannte Schönheitsfehler behoben.

	kundennr	Stundensumme
1	BEKL0001	414
2	HHHE0001	275

Abbildung 4.8: Verwendung eines Alias

Die Reihenfolge der Spaltenangaben in der SELECT-Anweisung wird bei der Ausgabe der Ergebnismenge berücksichtigt. Sie können die Reihenfolge beliebig festlegen und die Spaltenüberschrift mit einem *Aliasnamen* definieren.

4.4.2 Update

Mit der UPDATE-Anweisung werden vorhandene Daten einer Tabelle verändert. Es können eine, mehrere oder alle Zeilen einer Tabelle betroffen sein.

Syntax:

```
UPDATE table
SET column = bedingungsausdruck
  [ , column = bedingungsausdruck ] ...
  [ WHERE condition ]
```

oder

```
UPDATE table
SET ( column [ , column ] ... ) = ( Query)
 [ WHERE condition ]
```

Die einzelnen Komponenten der UPDATE-Anweisung haben ähnliche Funktionalität wie sie bereits bei der SELECT-Anweisung beschrieben wurden, trotzdem sollen sie nochmals kurz beschrieben werden.

Beschreibung:

UPDATE
Schlüsselwort für die UPDATE-Anweisung.

table
Tabelle, die von den Veränderungen betroffen ist.

SET
Schlüsselwort, welches die eigentliche Änderung einleitet.

column
Spalte(n), die von der Veränderung betroffen ist.

bedingungsausdruck
Neuer zugewiesener Wert.

[WHERE condition]
optionale Bedingung, welche die Auswirkung auf die betroffene Tabelle einschränkt.

Beispiel 4.22:

```
UPDATE adresse
SET plz = '40032'
WHERE plz = '4032';
```

In dem Beispiel 4.22 wird ein Fehler bei der Eingabe der Postleitzahlen korrigiert. Eine vierstellige Postleitzahl in der Tabelle *adresse* wird in eine fünfstellige Postleitzahl geändert.

Es können statt einem Wert, der neu zugewiesen werden soll, auch Abfragen verwendet werden, bei denen die Ergebnismengen den neuen Wertebeitrag liefern.

Beispiel 4.23:

```
UPDATE auftrag
SET anzstunden = (SELECT SUM(anzstunden) FROM service WHERE auftragnr = 6)
WHERE auftragnr = 6;
```

Die SQL-Anweisung in dem Beispiel 4.23 überträgt die Summe der angefallenen Stunden für einen Auftrag aus der Tabelle *service* in die Tabelle *auftrag*. In der SELECT-Anweisung wird mit der bereits kennen gelernten Funktion SUM die Anzahl der Stunden für einen Auftrag kummuliert und an die UPDATE-Anweisung übergeben. Die SELECT-Anweisung liefert nur einen Wert nämlich die ermittelte Summe. Dies muss bei der Erstellung der Anweisung berücksichtigt werden, denn bei mehreren Rückgabewerten würde es zu einem Fehler kommen, da nur ein Wert in der SET-Klausel erwartet wird. Die WHERE-Bedingung ist sowohl in der SELECT-Abfrage als auch in der UPDATE-Anweisung erforderlich, damit die Summe richtig berechnet wird und schließlich an der richtigen Stelle in der Tabelle *auftrag* geschrieben wird.

4.4.3 Insert

Damit vorhandene Daten verändert werden können, müssen sie zunächst auch irgendwie in die Tabellen gelangen. Dies geht bei einigen DBMS auch grafisch, der Weg des Programmierers geht jedoch über die INSERT-Anweisung. Diese Anweisung fügt einer Tabelle Datensätze hinzu. Es können entweder einzelne Datensätze (VALUE) oder auch mehrere, die über Abfragen bestimmt werden, eingefügt werden.

Syntax*:*

```
INSERT INTO table [ (column [ , column ] ... ]
{ VALUES ( value [ , value ] ... ) | Query }
```

Beschreibung:

INSERT INTO
Schlüsselwörter für die INSERT-Anweisung.

table
Tabelle, in die ein Datensatz oder mehrere eingefügt werden.

column
Angabe der Spalten, in die Werte geschrieben werden sollen (müssen von der Anzahl und der Reihenfolge mit den Angaben bei VALUES übereinstimmen).

VALUES
Schlüsselwort für die Klausel mit den Werten.

value
Angabe der Werte, die den benannten Spalten (bei Column) zugeordnet werden (müssen von der Anzahl und der Reihenfolge mit den Angaben bei Column übereinstimmen).

query

Anstatt einer Werteauflistung kann auch eine Unterabfrage die einzufügenden Werte liefern.

Eine praktische Anwendung dieses Befehls könnte das Hinzufügen eines Datensatzes in die Tabelle *telekom* sein.

Beispiel 4.24:

```
INSERT INTO telekom (adressnr, kategorienr, kategorie, kurzname, nummer)
VALUES (5, 4, 'Mobil','Klaus','0177-582733');
```

	telekomnr	recordstatus	adressnr	kategorienr	kategorie	kurzname	nummer
1	7	1	1	3	Telefon	Meyer	0221-56346
2	8	1	2	3	Telefon	Wühlbeck	0511-452310
3	9	1	5	3	Telefon	Schmidt	06321-98765
4	10	1	12	3	Telefon	Hinz	05321-23411
5	12	1	1	4	Fax	Meyer	0221-56347
6	13	1	5	3	Mobil	Schmidt	0171-98765
7	14	1	2	3	Fax	Wühlbeck	0511-452311
8	15	NULL	5	4	Mobil	Klaus	0177-582733

Abbildung 4.9: Datensätze nach einer INSERT-Anweisung

Dem kritischen Betrachter blieb sicherlich nicht verborgen, dass nicht alle möglichen Spalten bei der INSERT-Anweisung berücksichtigt wurden. Das ist auch kein Muss. Allerdings müssen all die Spalten einen Wert zugewiesen bekommen, bei den dies per Definition auch gefordert ist. Wäre bei unserem Beispiel in der Spalte *recordstatus* kein NULL-Wert erlaubt, so würde die INSERT-Anweisung, wie sie oben angeführt ist, eine Fehlermeldung produzieren. Da dies jedoch nicht der Fall ist, kann die Angabe für dieses einfache Testbeispiel vernachlässigt werden. Das Gleiche gilt für die Spalte *adressnr* und *telekomnr*, wobei letztere vom DBMS ihren Wert zugewiesen bekommt. Eine typische Fehlerquelle verbirgt sich auch bei der Auflistung der Werte. Erstens ist es notwendig sich über den zulässigen Wertebereich im Klaren zu sein, d.h. bei einem Character-Typ darf kein Integer-Wert zugewiesen werden. Zweitens kann sehr schnell ein Wert vergessen oder vertauscht werden und somit ist die ganze INSERT-Anweisung nicht ausführbar. Werden mit einer INSERT-Anweisung sämtliche Spalten gefüllt, dann müssen diese nicht zuvor aufgelistet werden. Es wäre also auch eine Anweisung wie die folgende möglich:

Beispiel 4.25:

```
INSERT INTO telekom
VALUES (1,5, 4, 'Fax','Klaus','03561-642376', 'AH1', 'AH1', '11.12.2001',
'11.12.2001');
```

Mit einer integrierten Abfrage können auch mehrere Datensätze gleichzeitig in eine Tabelle eingefügt werden. Hierbei müssen ebenso die Anzahl, Reihenfolge und Datentypen der Ergebnismenge aus der Abfrage mit denen der in der INSERT-Anweisung aufgeführten Spalten übereinstimmen.

Beispiel 4.26:

```
INSERT INTO telekom
SELECT * FROM telekom2;
```

4.4.4 Delete

Sollen Daten in einer Tabelle gelöscht werden, so kann dies mit der SQL-Anweisung DELETE geschehen. Im Gegensatz zur DROP-Anweisung wird hier keine Änderung an der Struktur vorgenommen.

Syntax:

```
DELETE FROM table [ WHERE condition ]
```

Beschreibung:

DELETE FROM
: Schlüsselbegriffe für die DELETE-Anweisung.

table
: Tabelle, in der Daten gelöscht werden sollen.

WHERE condition
: Optionale Bedingung, welche die Auswirkung auf die betroffene Tabelle einschränkt.

Ohne WHERE-Zusatz werden alle Datensätze einer Tabelle gelöscht. Die Telefonnummer eines Kunden ist nicht mehr gültig und muss deshalb aus der Datenbank gelöscht werden. Eine entsprechende DELETE-Anweisung könnte wie in Beispiel 4.27 aussehen.

Beispiel 4.27:

```
DELETE FROM telekom WHERE adressnr = 3 AND kategorienr = 5;
```

Der markierte Datensatz aus Abbildung 4.10 wird durch Ausführen der SQL-Anweisung gelöscht. In der Regel sind nur einzelne Zeilen zu löschen. In diesen Fällen werden in der WHERE-Klausel eine oder mehrere Bedingungen festgelegt.

telekomnr	adressnr	kategorienr	kategorie	kurzname	nummer
7	1	3	Telefon	Meyer	0221-56346
8	2	3	Telefon	Wühlbeck	0511-452310
9	3	3	Telefon	Schmidt	06321-98765
10	12	3	Telefon	Hinz	05321-23411
12	1	4	Fax	Meyer	0221-56347
13	3	5	Mobil	Schmidt	0171-98765
14	2	4	Fax	Wühlbeck	0511-452311
15	5	5	Mobil	Klaus	0177-582733
16	5	4	Fax	Klaus	03561-342376
17	4	3	Telefon	Heiner	0221-3552
18	12	5	Mobil	Hinz	0173-998

Abbildung 4.10: Datensätze in der Tabelle telekom vor dem Löschen

	telekomnr	recordstatus	adressnr	kategorienr	kategorie	kurzname	nummer
1	7	1	1	3	Telefon	Meyer	0221-56346
2	8	1	2	3	Telefon	Wühlbeck	0511-452310
3	9	1	3	3	Telefon	Schmidt	06321-98765
4	10	1	12	3	Telefon	Hinz	05321-23411
5	12	1	1	4	Fax	Meyer	0221-56347
6	14	1	2	4	Fax	Wühlbeck	0511-452311
7	15	1	5	5	Mobil	Klaus	0177-582733
8	16	1	5	4	Fax	Klaus	03561-342376
9	17	1	4	3	Telefon	Heiner	0221-3552
10	18	1	12	5	Mobil	Hinz	0173-998

Abbildung 4.11: Datensätze in der Tabelle telekom nach dem Löschen

4.5 DCL-Kontrollanweisungen

Jedes Datenbankobjekt wird durch den Eigentümer (Owner) erzeugt und dieser kann die verschiedenen Objekte für Benutzer oder auch Gruppen von Benutzern freigeben. So können die Benutzer auch mit Daten operieren, die sie nicht selbst besitzen, für die sie aber ein Zugriffsrecht haben. Die Mechanismen und Regeln für die Zugriffsrechte gehören auch zu dem Bereich der Datensicherheit. SQL bietet mit zwei Anweisungen `GRANT` und `REVOKE` die Mittel, mit denen Benutzer einander Rechte zuteilen und auch wieder entziehen können.

4.5.1 Zugriffsrechte erteilen

Objekberechtigungen steuern den Zugriff auf Objekte des DBMS. Damit ein Benutzer überhaupt einen Datensatz aus der DB lesen kann, muss ihm zuvor das Recht hierfür erteilt werden. Mit der GRANT-Anweisung lassen sich Rechte für Tabellen, Sichten, gespeicherte Prozeduren oder anderen Objekten erteilen.

Syntax:

```
GRANT { rechte | ALL PRIVILEGES}
ON objekt TO benutzer [,...n] [WITH GRANT OPTION]
```

Beschreibung:

GRANT
Schlüsselwort für die SQL-Anweisung.

rechte
Recht oder Liste mit Berechtigungen, die für das benannte Objekt möglich sind. Mit ALL PRIVILEGES können alle Rechte für dieses Objekt vergeben werden.

objekt
Objekt, auf welches die vergebenen Rechte zutrifft.

benutzer
Ein oder mehrere durch Komma getrennte Benutzer, denen Rechte an den genannten Objekten erteilt wird.

WITH GRANT OPTION
Es wird dem Benutzer erlaubt, die erhaltenen Rechte auch auf andere Benutzer zu übertragen.

Beispiel 4.28:

```
GRANT SELECT, INSERT
ON telekom TO aherbolsheimer WITH GRANT OPTION;
```

Welche Auswirkung hat nun die Anweisung im Beispiel 4.28? Dem Benutzer *aherbolsheimer*, welcher zuvor angelegt sein muss, wird die Berechtigung erteilt auf die Tabelle *telekom* lesend und schreibend zuzugreifen. Diese Berechtigung kann der Benutzer auch an andere weitergeben. Je nach Vertrauenslage in einem Unternehmen, sollte die `WITH GRANT OPTION` Klausel jedoch mit äußerster Vorsicht verwendet werden, weil damit der Administrator ein Stück Kontrollfunktion abgibt.

4.5.2 Zugriffsrechte entziehen

Bereits erteilte Rechte lassen sich aber auch sehr schnell wieder entziehen, ohne dass dabei der Benutzer gelöscht und wieder angelegt werden müsste. Mit der REVOKE-Operation können die Rechte, die der Benutzer an einem Objekt genießt, wieder genommen werden.

Syntax:

```
REVOKE [GRANT OPTION FOR] { rechte | ALL PRIVILEGES}
ON objekt FROM benutzer [,...n] {RESTRICT | CASCADE}
```

Beschreibung:

REVOKE
Schlüsselwort für die SQL-Anweisung.

GRANT OPTION FOR
Berechtigung zur Rechtevergabe wird entzogen.

rechte
Recht oder Liste mit Berechtigungen, die für das benannte Objekt möglich sind. Mit ALL PRIVILEGES können alle Rechte für dieses Objekt entzogen werden.

objekt
Objekt, auf welches die entzogenen Rechte zutrifft.

benutzer
Ein oder mehrere durch Komma getrennte Benutzer, denen Rechte an den genannten Objekten entzogen werden.

RESTRICT | CASCADE
Die Weitergabe von Berechtigungen wird ebenfalls rückgängig gemacht.

Beispiel 4.29:

```
REVOKE Insert ON telekom FROM aherbolsheimer CASCADE;
```

In dem angegebenen Beispiel wird die vorher erteilte Berechtigung für den Benutzer *aherbolsheimer* wieder eingeschränkt. Das Recht auf die Ausführung eines Insert-Befehls wird genommen und durch die Angabe `CASCADE` werden auch den Benutzern diese Berechtigung genommen, die mittlerweile vom Benutzer *aherbolsheimer* diese Berechtigung bekommen haben. Der Unterschied zwischen `RESTRICT` und `CASCADE` liegt in der Ebene der zuvor erhaltenen Berechtigung.

Beispiel 4.30:

Benutzer B1 erteilt Benutzer B2 das Recht auf ein Objekt zuzugreifen und dieses Recht auch weiterzugeben. Daraufhin erteilt B2 auch Benutzer B3 dieses Recht. Mit der Angabe von CASCADE in der REVOKE-Anweisung würde Benutzer B1 nun beiden, sowohl B2 als auch B3, dieses Recht wieder nehmen können. Die Rechte werden herunter kaskadiert entzogen. Benutzer B2 müsste jedoch, um Benutzer B3 (und weiteren Benutzern) dieses Recht zu entziehen, in der REVOKE-Anweisung mit der RESTICT-Klausel arbeiten.

Je nach RDBMS werden für die Rechteverwaltung auch Systemprozeduren eingesetzt oder die Verwaltung erfolgt auf dem grafischen Weg, so wie dies optional auch beim MS SQL Server möglich ist (siehe Abbildung 4.12).

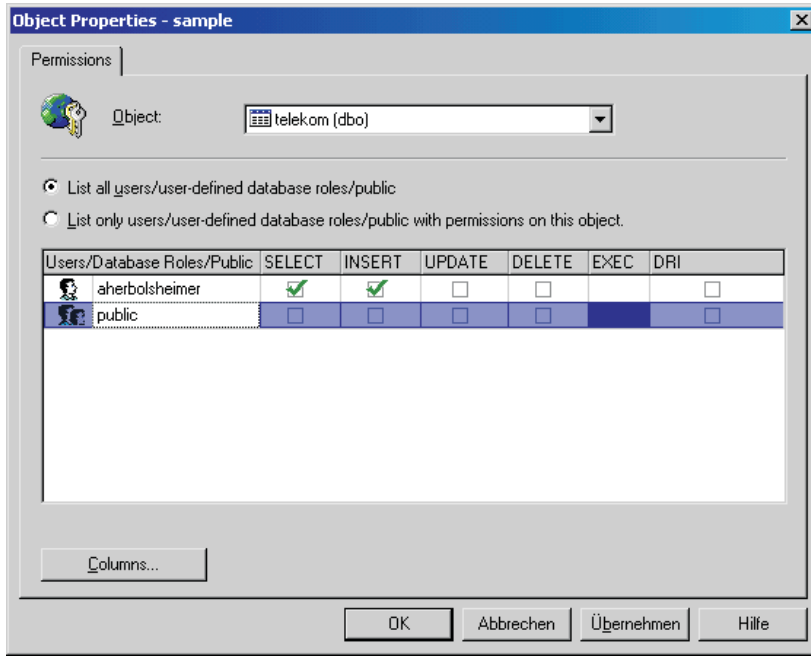


Abbildung 4.12: Berechtigungsvergabe beim MS SQL Server

Bei MS Access hält diese *Berechtigungsvergabe* auf Objektebene Einzug in das Sicherheitssystem ab der Version 2000. Beim Arbeiten mit PostgreSQL können sie wie oben gezeigt mit den entsprechenden SQL-Anweisungen die Vergabe von Berechtigungen kontrollieren. Bei PostgreSQL und auch beim MS SQL Server können zusätzlich *Gruppen* definiert werden. Gruppen kann man als Zusammenfassungen von Benutzern verstehen, die mit gleicher Berechtigung ausgestattet sind.

4.6 Erweiterte Abfragetechniken

Die Grundlagen der SQL-Anweisungen wurden bereits behandelt. Mit den kennen gelernten Befehlen kann man schon einiges ausrichten beim Umgang mit relationalen Datenbanken. Bei der tägliche Arbeit an und mit Datenbanken wird sich jedoch zeigen, dass gerade bei der Selektion der Daten viele Wünsche offen bleiben. Schon bald möchte man nicht mehr nur Daten aus einer sondern aus mehreren Tabellen als Ergebnismenge. Außerdem wollen wir zusammengefasste und sortierte Informationen als

Ergebnis. Vielleicht interessiert uns, welcher Kunde im laufenden Geschäftsjahr die meisten Aufträge erteilt hat und welcher Kunde an zweiter, dritter, vierter usw. Stelle steht. Die Techniken für die Beantwortung dieser und weiterer Fragestellungen werden wir in diesem Abschnitt kennen lernen.

4.6.1 Daten aus mehreren Tabellen

Bei der Vorstellung des relationalen Modells wurde als typisches Kennzeichen die Verteilung der Daten auf mehrere Relationen (Tabellen) genannt. Weil Operationen und daraus resultierende Ergebnismengen häufig mehrere Tabellen betreffen, müssen diese über Spalten, die in beiden (oder mehreren) Tabellen enthalten sind, verknüpft werden. In Anlehnung an die Ursprünge aus der Relationenalgebra wird diese Verknüpfung auch oft Verbund genannt. Es gibt drei Verknüpfungstypen:

- Innere Verknüpfung,
- Äußere Verknüpfung,
- Kreuzverknüpfung.

Zunächst soll die erweiterte Syntax der SELECT-Anweisung vorgestellt werden, da sie Ausgangspunkt für alle Abfragen mit den drei Verknüpfungstypen ist.

Syntax:

```
SELECT [ ALL | DISTINCT ] { * | table.* | bedingungsausdruck } [ , { * | table.* |
bedingungsausdruck } ] ...
FROM <table_source>
[ WHERE bedingungsausdruck ]
[ GROUP BY spalte_kommaliste [ HAVING bedingungsausdruck ] ]
[ ORDER BY { bedingungsausdruck |position } [ ASC |DESC ] ... ]
table_source
::= tabellenausdruck_mit_join | tabellenausdruck_ohne_join

tabellenausdruck_mit_join
::= tabellenreferenz CROSS JOIN tabellenreferenz | tabellenreferenz [NATURAL]
[verbundtyp] JOIN tabellenreferenz [ ON bedingungsausdruck | USING
(spalte_kommaliste)] | (tabellenausdruck_mit_join)

tabellenreferenz
::= tabellenausdruck_mit_join | tabelle [ [ AS ] bereichsvariable [
(spalte_kommaliste)] ]
|  (tabellenausdruck) [ AS ] bereichsvariable [ (spalte_kommaliste) ]

verbundtyp
::=  INNER
     | LEFT [ OUTER ]
     | RIGHT [ OUTER ]
```

```
      | FULL [ OUTER ]
      | UNION

tabellenausdruck_ohne_join
::= tabellenterm_ohne_join | tabellenausdruck { UNION | EXCEPT } [ ALL ]
[ CORRESPONDING [BY (spalte_kommaliste) ] ] tabellenterm
```

Beschreibung:

CROSS JOIN
Schlüsselbezeichnung, die eine Kreuzverknüpfung einleitet.

tabellenreferenz
Name einer Tabelle, welche bei der Verknüpfung beteiligt ist.

NATURAL
Kennzeichnung eines natürlichen Verbundes. Es wurden weder eine USING- noch eine ON-Klausel angegeben.

Verbundtyp
Typus der verwendeten Verknüpfung.

AS bereichsvariable
Definition eines Aliasnamens zur Unterscheidung der an der Verknüpfung beteiligten Spalten und Tabellen.

USING
In der USING-Klausel wird eine Liste mit Spalten angegeben, in der die Spalten der zu verknüpfenden Tabellen stehen. Es wird entweder die ON-Klausel oder die USING-Klausel verwendet.

UNION
Es wird eine Vereinigungsmenge aus den verknüpften Tabellen gebildet.

EXCEPT
Es wird die Differenzmenge aus den verknüpften Tabellen gebildet.

CORRESPONDING
Schlüsselwort, dem eine korrespondierende Tabelle bzw. die Spalten der Tabelle folgen.

Einige Beispiele sollen die Funktionsweise der Verknüpfungstypen und die Bedeutung der syntaktischen Teilbereiche aufzeigen.

Innere Verknüpfung

Zunächst soll eine SELECT-Anweisung eine Verknüpfung zweier Tabellen mit `INNER JOIN` herstellen. Mit inneren Verknüpfungen werden Tabellen kombiniert, indem Werte in Spalten verglichen werden, die in beiden Tabellen vorkommen. Es werden die Datensätze zurückgegeben, die den Verknüpfungsbedingungen entsprechen.

Beispiel 4.31:

```
SELECT adresse.name,telekom.kategorie,telekom.nummer
FROM adresse INNER JOIN telekom
ON adresse.adressnr = telekom.adressnr;
```

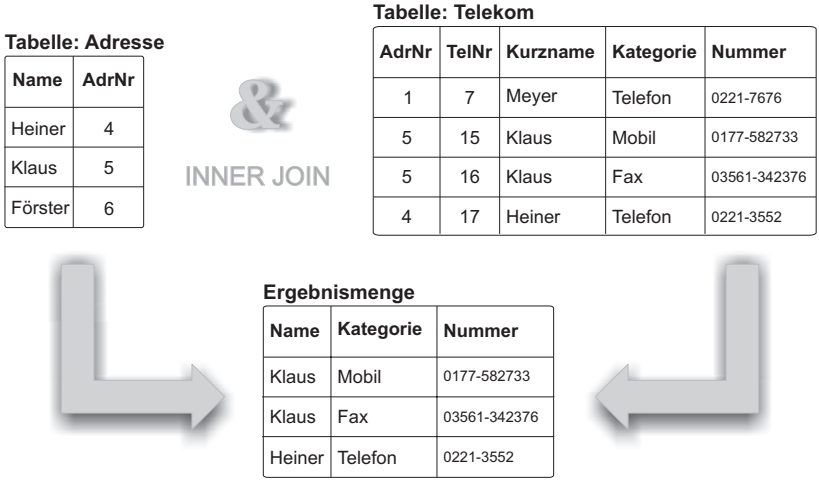


Abbildung 4.13: Beschreibung der inneren Verknüpfung

Die Abbildung 4.13 zeigt die Ausgangstabellen, die über eine innere Verknüpfung die angegebene Ergebnismenge liefern. Es erscheinen nur die Datensätze, die der Verknüpfungsbedingung entsprechen, d.h. die *adressnr* aus *telekom* muss gleich der *adressnr* aus *adresse* sein. Der Datensatz aus der Tabelle *adresse* mit der adressnr = 6 wird nicht berücksichtigt, weil kein referenzierter Datensatz in der Tabelle *telekom* vorhanden ist. Ebenso wird die Zeile mit der telekomnr = 7 aus der Tabelle Telekom nicht berücksichtigt, weil sie keine Referenz in der Tabelle *adresse* hat. Zu beachten ist der Umstand, dass in der ON-Klausel die Spaltenangaben zusätzlich mit der Tabellenbezeichnung versehen sind. Andernfalls hätte man an dieser Stelle eine Mehrdeutigkeit, die vom DBMS nicht ausgewertet werden kann. Das Gleiche gilt für die Liste der Spalten nach dem Schlüsselwort `SELECT`, die bei mehrfachem Vorkommen ebenfalls mit der Tabellenbezeichnung angegeben werden müssen. Die Ergebnismenge kann durch eine zusätzliche WHERE-Bedingung weiter eingeschränkt werden, wie das folgende Beispiel zeigt:

Beispiel 4.32:

```
SELECT adresse.name,telekom.kategorie,telekom.nummer
FROM adresse INNER JOIN telekom
ON adresse.adressnr = telekom.adressnr
WHERE telekom.kategorie = 'Telefon';
```

name	kategorie	nummer
Heiner	Telefon	0221-3552

Abbildung 4.14: Ergebnis der SELECT-Abfrage aus Beispiel 4.32

Durch die WHERE-Bedingung werden zwei Datensätze herausgefiltert und nur der Datensatz wird ausgewiesen, der die Kategorie »Telefon« hat. Auf diese Weise können Informationen ganz gezielt ausgewertet werden und auch für Zwischenschritte ist es vorteilhaft, nur mit den wirklich relevanten Daten zu operieren. Bei einigen DBMS ist die *innere Verknüpfung* die Standardverknüpfung und die INNER JOIN-Klausel kann deshalb mit `JOIN` abgekürzt werden. Es sollten keine Nullwerte als Verknüpfungsbedingung verwendet werden, da Nullwerte nicht übereinstimmend ausgewertet werden.

Äußere Verknüpfung

Bei der *äußeren Verknüpfung* wird unterschieden zwischen der linken und der rechten äußeren Verknüpfung. Mit linken oder rechten äußeren Verknüpfungen werden Zeilen aus zwei Tabellen kombiniert, die der Verknüpfungsbedingung entsprechen, sowie in der JOIN-Klausel angegebene Zeilen ohne Übereinstimmung aus der linken oder rechten Tabelle. In Zeile, die der Verknüpfungsbedingung nicht entsprechen, wird in der Ergebnismenge NULL angezeigt. Vollständige äußere Verknüpfungen dienen zur Anzeige aller Zeilen der verknüpften Tabellen, unabhängig davon, ob in den Tabellen übereinstimmende Werte vorhanden sind.

Beispiel 4.33:

```
SELECT adresse.name,telekom.kategorie,telekom.nummer
FROM adresse LEFT OUTER JOIN telekom
ON adresse.adressnr = telekom.adressnr;
```

Das Beispiel von oben wurde nur geringfügig modifiziert. Genau genommen wurde hier statt eines `INNER JOIN` ein `LEFT OUTER JOIN` verwendet. Die Ausgangstabellen bleiben dieselben und die Verknüpfungsbedingung hat sich ebenfalls nicht geändert. Das Ergebnis sehen wir in Abbildung 4.15. Es ist eine weitere Zeile hinzugekommen, die aus der linken Tabelle *adresse* resultiert und bei den Spalten aus der Tabelle *telekom* NULL-Werte aufweist.

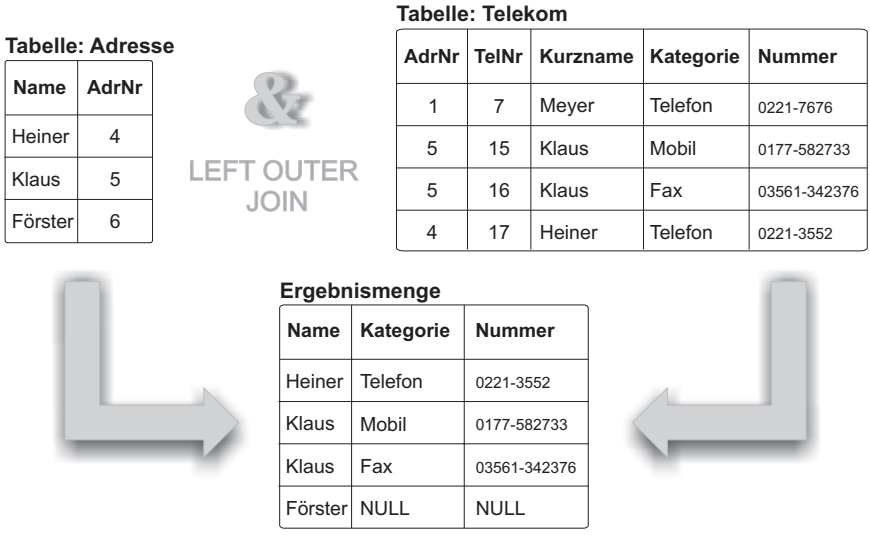


Abbildung 4.15: Beschreibung zur äußeren Verknüpfung

Das gleiche Beispiel mit einer *rechten äußeren Verknüpfung* bringt uns ein anderes Ergebnis. Diese Ergebnismenge könnte eventuell bei einer Suche nach »toten« Datensätzen gebraucht werden.

Beispiel 4.34:

```
SELECT adresse.name,telekom.kategorie,telekom.nummer
FROM adresse RIGHT OUTER JOIN telekom
ON adresse.adressnr = telekom.adressnr;
```

name	kategorie	nummer
NULL	Telefon	0221-56346
NULL	Fax	0221-56347
Klaus	Mobil	0177-582733
Klaus	Fax	03561-342376
Heiner	Telefon	0221-3552

Abbildung 4.16: Ergebnis der Anweisung mit äußeren Verknüpfungen

Die äußere Verknüpfung sollte verwendet werden, wenn neben den Daten eine vollständige Liste der Daten benötigt wird. Die *linke* und *rechte äußere Verknüpfung* beziehen sich auf die Reihenfolge, in der die Tabellen in FROM-Klausel angegeben wurden. Werden die Tabellen dort vertauscht, so bekommt man bei einer *linken äußeren Verknüpfung* das gleiche Ergebnis, wie wenn man eine *rechte äußere Verknüpfung* mit der ursprünglichen Reihenfolge gewählt hat. Auch hier können Nullwerte als Verknüp-

fungsbedingung nicht ausgewertet werden. Eine Vereinfachung in der JOIN-Klausel in der Form

```
LEFT OUTER JOIN -> LEFT JOIN
RIGHT OUTER JOIN -> RIGHT JOIN
```

ist zulässig.

Kreuzverknüpfung

Bei der *Kreuzverknüpfung* werden alle möglichen Kombinationen sämtlicher Zeilen der verknüpften Tabellen angezeigt. Für die Verwendung der Kreuzverknüpfung ist keine Verknüpfungsbedingung erforderlich. Die Anzahl der Zeilen in der Ergebnismenge entspricht dem *kartesischen Produkt* aus der Zeilenanzahl der ersten Tabelle multipliziert mit der Anzahl der Zeilen aus der zweiten Tabelle.

Beispiel 4.35:

```
SELECT adresse.name,telekom.kategorie,telekom.nummer
FROM adresse CROSS JOIN telekom;
```

name	kategorie	nummer
Heiner	Telefon	0221-56346
Heiner	Mobil	0177-582733
Heiner	Fax	03561-342376
Heiner	Telefon	0221-3552
Klaus	Telefon	0221-56346
Klaus	Mobil	0177-582733
Klaus	Fax	03561-342376
Klaus	Telefon	0221-3552
Förster	Telefon	0221-56346
Förster	Mobil	0177-582733
Förster	Fax	03561-342376
Förster	Telefon	0221-3552

Abbildung 4.17: Ergebnismenge bei der Kreuzverknüpfung

Das Ergebnis zeigt, dass die Daten faktisch nicht mehr richtig zugeordnet werden. Deshalb wird die Kreuzverknüpfung in normalisierten Datenbanken üblicherweise nur zum Erzeugen von Testdaten verwendet.

Verknüpfung von drei beteiligten Tabellen

Manchmal erfordert es die Anwendung, dass nicht nur zwei sondern mehrere Tabellen miteinander verknüpft werden. Grundsätzlich kann eine beliebige Anzahl von Tabellen miteinander verknüpft werden. Über eine gemeinsame Spalte ist die Verknüpfung jeder Tabelle, auf die in einer Verknüpfungsoperation verwiesen wird, mit einer anderen Tabelle möglich.

Beispiel 4.36:

```
SELECT adresse.name, telekom.kategorie, telekom.nummer, kunde.KmEntfernungen
FROM adresse  JOIN telekom
ON adresse.adressnr = telekom.adressnr
JOIN kunde
ON adresse.adressnr = kunde.adressnr;
```

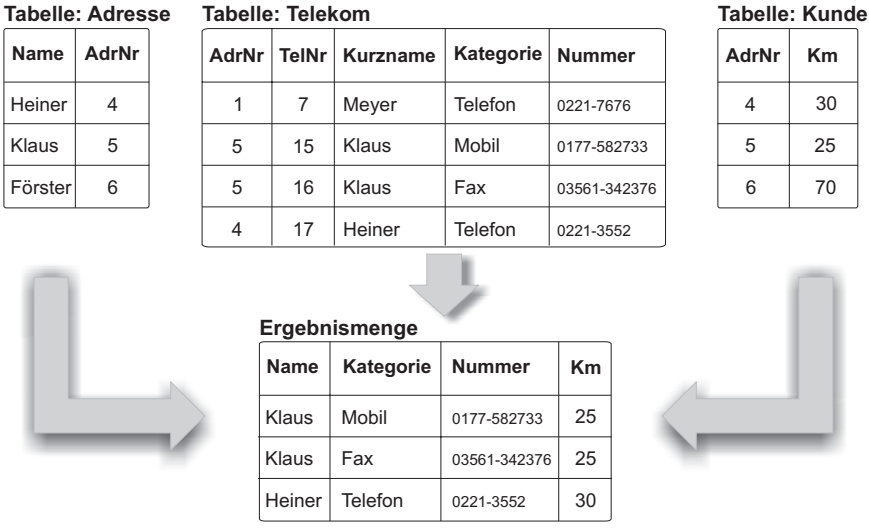


Abbildung 4.18: Beschreibung zur Verknüpfung mehrerer Tabellen

Die Abbildung 4.18 zeigt uns auch die Ergebnismenge dieser SQL-Anweisung. Die Spalte *km* aus der Tabelle *kunde* wird der Ergebnismenge aus Abbildung 4.13, wo es um die Verknüpfung der Tabellen *adresse* und *telekom* ging, zugefügt.

Verknüpfungen von mehreren Tabellen sind also möglich, wenn die Tabellen Fremdschlüsselbeziehungen zu jeder Tabelle enthalten, die verknüpft werden. Für jede Spalte, die Teil eines zusammengesetzten Schlüssels ist, ist eine JOIN-Klausel erforderlich. Es können zwar beliebig viele Tabellen miteinander verknüpft werden, aber eine hohe Anzahl an Verknüpfungen geht zu Lasten der steigenden Komplexität und geringeren *Abfragegeschwindigkeit*.

Verknüpfung einer Tabelle mit sich selbst

Es kommt in der Praxis nicht häufig vor, dass man eine Tabelle mit sich selbst verknüpfen muss, aber es kann eine Hilfe bei der Datenbankanalyse sein. Wenn Zeilen gesucht werden, deren Wert mit den Werten in anderen Zeilen derselben Tabelle übereinstimmen, kann diese Tabelle über eine Selbstverknüpfung diese Zeilen ausweisen. Bei einer Selbstverknüpfung der Tabelle müssen *Aliasnamen* für Tabelle und Spalte vergeben werden. Die Tabellen-Aliasnamen müssen sich von den Spalten-Aliasnamen unterscheiden. Bei einer einfachen Selbstverknüpfung würde nun jede Zeile mit sich selbst übereinstimmen und dadurch würde es zu einer Paarbildung in der Ergebnismenge kommen. Mit einer WHERE-Bedingung können diese Zeilen mit *Redundanz* herausgefiltert werden.

Beispiel 4.37:

```
SELECT a.telekomnr as telekom1, a.adressnr, b.telekomnr as telekom2
FROM telekom a  JOIN telekom b
ON a.adressnr = b.adressnr
WHERE a.telekomnr > b.telekomnr;
```

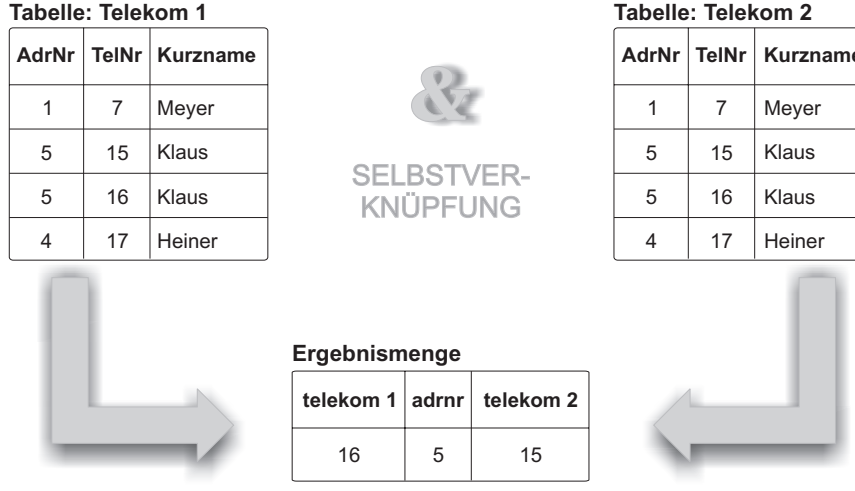


Abbildung 4.19: Verknüpfung einer Tabelle mit sich selbst

Das angegebene Beispiel sucht in der Tabelle *telekom* mehrere Datensätze zu einer *adressnr*, d.h. es werden Personen gesucht, die mehr als eine Telekommunikationsangabe gemacht haben.

4.6.2 Arbeiten mit Aggregatfunktionen

Funktionen, die Durchschnittswerte und Summen berechnen, werden Aggregatfunktion genannt. Eine Übersicht der zur Verfügung stehenden Aggregatfunktionen nach ANSI wurde bereit in Tabelle 4.6 gegeben. Wird eine Aggregatfunktion ausgeführt, fasst das DBMS Werte für eine gesamte Tabelle oder für Gruppen von Spalten in der Tabelle zusammen. Aggregatfunktionen können mit der SELECT-Anweisung oder in Kombination mit der GROUP BY-Klausel verwendet werden. Mit Ausnahme der COUNT(*)-Funktion geben alle Aggregatfunktionen NULL zurück, wenn keine Zeile die WHERE-Klausel erfüllt. Die COUNT(*)-Funktion gibt den Wert Null zurück, wenn keine Zeile die WHERE-Klausel erfüllt. Außer im Fall von COUNT(*) kann dem Ausdruck optional das Schlüsselwort `DISTINCT` vorangestellt sein, um redundante Werte vor Anwendung der Funktion entfernt werden. Je nach Datentyp der Spalten können Aggregatfunktionen ausgeführt werden oder nicht. So muss für `SUM` und `AVG` das Argument von einem numerischen Typ sein. Zeichenkettenausdrücke können nur von der Funktion `COUNT` verwendet werden.

Beispiel 4.38:

```
SELECT COUNT(*) AS Anzahl
FROM telekom
WHERE kategorie = 'Telefon';
```

Anzahl
5

Abbildung 4.20: Ergebnismenge zum Beispiel 4.38

Das Beispiel zeigt eine SQL-Anweisung, mit der die Anzahl der Datensätze in der Tabelle *telekom* ermittelt wird, welche eine Telefonnummer beinhalten. Die Ergebnismenge umfasst nur eine Spalte, deren Bezeichnung mit dem Aliasnamen *Anzahl* festgelegt wurde.

Beispiel 4.39:

```
SELECT COUNT(DISTINCT adressnr) AS Anzahl
FROM telekom;
```

Das Beispiel zählt die Adressnummern in der Tabelle *telekom*. Dabei werden Duplikate nicht mitgezählt, weil diese durch die DISTINCT-Anweisung unmittelbar vor der Spaltenbezeichnung verhindert wird. Ohne DISTINCT wird der Wert in der Ergebnismenge höher sein.

Im nächsten Beispiel soll der Kunde herausgefunden werden, der am weitesten (räumliche Distanz) von der jeweiligen Niederlassung, die ihn betreut, entfernt ist.

Beispiel 4.40:

```
SELECT kundennr
FROM Kunde
WHERE kmentfernungen=
(SELECT MAX(kmentfernungen) FROM Kunde);
```

In der Unterabfrage wird zunächst der Maximalwert für die Entfernung ermittelt. Das Ergebnis der Unterabfrage wird in der WHERE-Bedingung der Hauptabfrage ausgewertet. Schließlich wird die Nummer des Kunden mit der größten Entfernung als Ergebnismenge geliefert.

Die Kunden, die über dem Umsatzdurchschnitt der Kundschaft je Auftrag liegen, sollen aufgelistet werden. Dafür kann die folgende Anweisung verwendet werden.

Beispiel 4.41:

```
SELECT kundennr
FROM auftrag
WHERE (abgnebenkosten+abghonorar)>
(SELECT AVG(abgnebenkosten+abghonorar) FROM auftrag);
```

Das nächste Beispiel stellt die Gesamtbeträge für das Sollhonorar und das abgerechnete Honorar des Unternehmens gegenüber.

Beispiel 4.42:

```
SELECT SUM(abghonorar) as ist, SUM(sollhonorar) as soll
FROM auftrag;
```

Nun macht dies noch nicht viel Sinn, wenn diese Abfrage nicht zeitlich eingeschränkt wird. Im nächsten Beispiel sollen nur Aufträge berücksichtigt werden, die im Jahr 2000 abgerechnet wurden. In der WHERE-Bedingung wird deshalb das Rechnungsdatum überprüft. Die Funktion `YEAR` extrahiert die Jahresangabe aus dem angegebenen Datum.

Beispiel 4.43:

```
SELECT SUM(abghonorar) as ist, SUM(sollhonorar) as soll
FROM auftrag
WHERE YEAR(rechndatum)= '2000';
```

Spalten, die häufig für Aggregatfunktionen verwendet werden, sollten indiziert werden. Durch die Indizierung werden die Aggregatoperationen beschleunigt.

4.6.3 Zusammenfassen der Daten

Im Zusammenhang mit Aggregatfunktionen können Ergebnismengen mit der GROUP BY- und HAVING-Klausel zusammengefasst werden. `GROUP BY` ist optionaler Bestandteil des SELECT-Ausdrucks und wird verwendet, um Zeilen in Gruppen zu strukturieren und als solche zusammenzufassen.

GROUP BY-Klausel

Es sollen im Folgenden die Aufträge je Kunde im Geschäftsjahr 2000 gezählt und das Ergebnis gruppiert nach Kunde ausgegeben werden.

Beispiel 4.44:

```
SELECT kundennr AS Kundennummer, COUNT(auftragnr) AS Aufträge
FROM auftrag
WHERE year(rechndatum)= '2000'
GROUP BY kundennr;
```

Die Aufträge werden mir der Aggregatfunktion `COUNT` gezählt. Für die Spalte mit dem Zählergebnis wird der Aliasname *Aufträge* vorgesehen. Die zweite Spalte, die in der Ergebnismenge erscheinen soll ist die Kundennummer. Das Zählergebnis soll je Kunde zusammengefasst werden, darum wird diese Spalte in der GROUP BY-Klausel angeführt. Alle Spalten, die in der GROUP BY-Klausel angegeben sind, müssen Bestandteil der Auswahlliste ein. Die WHERE-Bedingung verhindert, dass Datensätze mitgezählt werden, die nicht der Bedingung entsprechen. Das Ergebnis zu diesem Beispiel finden wir in Abbildung 4.21.

Kundennummer	Aufträge
BEFH0002	1
BEKL0001	2
HHHE0001	2

Abbildung 4.21: SELECT-Abfrage mit GROUP BY-Klausel

Mit der ORDER BY-Klausel wird schließlich auch die Sortierreihenfolge vorgegeben, in der die Ergebniszeilen ausgegeben werden.

Beispiel 4.45:

```
SELECT kundennr AS Kundennummer,COUNT(auftragnr) AS Aufträge
FROM auftrag
WHERE year(rechndatum)= '2000'
GROUP BY kundennr
ORDER BY Aufträge DESC;
```

Der Kunde mit den meisten Aufträgen soll zuerst erscheinen und anschließend die anderen Kunden in absteigender Reihenfolge. Deshalb wird die Spalte Aufträge als Sortierkriterium verwendet und das Schlüsselwort `DESC` gibt die absteigende Reihenfolge an.

Beispiel 4.46:

```
SELECT kunde.kundennr, adresse.name, adresse.plz, COUNT(auftragnr) AS Aufträge
FROM adresse JOIN kunde ON adresse.adressnr = kunde.adressnr
JOIN auftrag ON kunde.kundennr= auftrag.kundennr
GROUP BY kunde.kundennr, adresse.name, adresse.plz;
```

Vielleicht werden für die einzelnen Kunden auch gleich die Namen und Postleitzahlen benötigt. Diese Angaben finden wir in der Tabelle *adresse*. Nun gibt es aber keinen direkten Bezug zwischen den Tabellen *adresse* und *auftrag*. Es gibt jedoch über die *adressnr* eine Schlüsselbeziehung zwischen *adresse* und *kunde*. Über die Tabelle *kunde* kann dann auch mit der Schlüsselbeziehung *kundennr* eine Verknüpfung mit der Tabelle *auftrag* hergestellt werden. Die Spalten, welche nicht aggregiert werden, müssen auch hier ausnahmslos in der GROUP BY-Klausel aufgelistet werden.

kundennr	name	plz	Aufträge
BEFH0002	Förster	20001	1
BEKL0001	Klaus	34550	3
HHHE0001	Heiner	40032	3

Abbildung 4.22: Ergebnis der Abfrage aus Beispiel 4.45

Die GROUP BY-Klausel kann bei Spalten, die mehrere NULL-Werte enthalten, nicht verwendet werden, da die NULL-Werte als Gruppe verarbeitet werden.

HAVING-Klausel

Einen weiteren Teil der SELECT-Anweisung lernen wir mit der HAVING-Klausel kennen. Diese kann nur in Verbindung mit einer GROUP BY-Anweisung angewendet werden. Mit der HAVING-Klausel wird der Ergebnismenge einer Abfrage mit GROUP BY-Ausdruck eine Bedingung angefügt, die sich auf die einzelnen Gruppen im Ergebnis bezieht. Die HAVING-Klausel kann mehrere Bedingungen enthalten, die mit logischen Operatoren kombiniert werden.

Beispiel 4.47:

```
SELECT kunde.kundennr, adresse.name, adresse.plz,  COUNT(auftragnr) AS Aufträge
FROM adresse JOIN kunde ON adresse.adressnr = kunde.adressnr
JOIN auftrag ON kunde.kundennr= auftrag.kundennr
GROUP BY kunde.kundennr, adresse.name, adresse.plz
HAVING COUNT(auftragnr) > 1;
```

kundennr	name	plz	Aufträge
BEKL0001	Klaus	34550	3
HHHE0001	Heiner	40032	3

Abbildung 4.23: SELECT-Abfrage mit HAVING-Klausel

Beim Vergleich der Ergebnisse aus Bild 4.22 und Bild 4.23 wird der Unterschied mit und ohne HAVING-Klausel deutlich. Die Bedingung, Anzahl der Aufträge muss größer als eins sein, lässt den Kunden mit der Nummer *BEFH0002* aus der Ergebnismenge fallen. Mit einer WHERE-Bedingung könnte man diesen Effekt nicht erreichen, weil sich die WHERE-Bedingung nicht auf die Ergebnismenge bezieht, sondern auf die Ausgangsdatensätze. Eine *Aggregatfunktion* darf also nicht direkt in der WHERE-Klausel verwendet werden. In der HAVING-Klausel ist dies jedoch erlaubt. Theoretisch kann auf jede Spalte der Auswahlliste verwiesen werden, was jedoch nicht immer sinnvoll ist.

Beispiel 4.48:

```
SELECT kundennr AS KundenNr, sum(abgnebenkosten) AS AbgNebenkosten,
sum(sollnebenkosten) AS SollNebenkosten, sum(abghonorar) AS Honorar
FROM auftrag
WHERE YEAR(rechndatum)= '2000'
GROUP BY kundennr
HAVING sum(abgnebenkosten) > sum(sollnebenkosten);
```

KundenNr	AbgNebenkosten	SollNebenkosten	Honorar
BEKL0001	3520.0000	550.0000	15740.0000
HHHE0001	2820.0000	100.0000	9199.0000

Abbildung 4.24: Kombination der WHERE-Klausel mit der HAVING-Klausel

Wie das angegebene Beispiel zeigt, können auch WHERE- und HAVING-Bedingung in Kombination in einer SQL-Anweisung auftreten. Hier werden die abgerechneten Nebenkosten den Sollkosten aus dem Geschäftsjahr 2000 gegenübergestellt und nach *KundenNr* gruppiert. Die Aufträge, bei denen die Nebenkosten-Summe im Soll höher ausfällt als die abgerechnete Summe der Nebenkosten, sollen dabei nicht Teil des Ergebnisses sein. Durch die Bedingung in der HAVING-Klausel werden diese Zeilen ausgeblendet.

4.6.4 Unterabfragen

Abfragen sind manchmal abhängig von Ergebnissen anderer Abfragen. Dies wurde schon deutlich am Beispiel mit der Kundenliste, bei der nur Kunden mit einem durchschnittlich höheren Betrag beim Umsatz, ausgewiesen werden sollten. Manchmal ist es auch gut, wenn eine komplexe Abfrage in mehrere logische Einzelschritte aufgegliedert wird. In beiden Fällen ist es denkbar, mit *Unterabfragen* zu arbeiten. Eine Unterabfrage ist eine SELECT-Anweisung, die geschachtelt innerhalb einer SELECT-, INSERT-, UPDATE-, oder DELETE-Anweisung auftritt.

Beispiel 4.49:

```
SELECT kundennr AS KundenNr, auftrdatum AS Auftragdatum
FROM auftrag
WHERE auftrdatum = (SELECT MAX(auftrdatum) FROM auftrag);
```

KundenNr	auftrdatum
BEFH0002	2000-12-15 00:00:00.000

Abbildung 4.25: Verwendung einer Unterabfrage

In dem Beispiel 4.49 wird die *Unterabfrage* verwendet, um das letzte Auftragsdatum zu ermitteln. Es wird ein Wert zurückgegeben, nämlich die Datumsangabe. In der *Hauptabfrage* werden schließlich alle Kunden über ihre Nummer ausgewiesen, die an dem Datum (aus der Unterabfrage) einen Auftrag erteilt haben.

Die Unterabfrage kann sowohl einen, als auch beliebig viele Rückgabewerte an die aufrufende Stelle übergeben. Die nächste SQL-Anweisung soll exemplarisch für eine Unterabfrage mit mehreren Rückgabewerten stehen.

Beispiel 4.50:

```
SELECT adressnr, name
FROM adresse
WHERE adressnr NOT IN (SELECT adressnr FROM telekom);
```

Die Abfrage liefert alle Datensätze der Tabelle *adresse*, die keinen referenzierten Datensatz in der Tabelle *telekom* haben. Die Unterabfrage hat in der Ergebnismenge alle Adressnummern aus der Tabelle *telekom*, diese werden mit den Adressnummern aus der Tabelle *adresse* verglichen.

Einen speziellen Fall stellen die *korrelierten Unterabfragen* dar. Bei der korrelierten Unterabfrage braucht die innere Abfrage Informationen aus der äußeren Abfrage zur Ausführung. Zur Unterscheidung der Tabellen müssen dann Aliasnamen verwendet werden.

Beispiel 4.51:

```
SELECT kundennr, name
FROM kunde AS k
WHERE EXISTS
(SELECT * FROM auftrag WHERE k.kundennr = auftrag.kundennr
AND YEAR(rechndatum)> '1999');
```

Die Beispielanweisung listet Name und Nummer der Kunden auf, welche noch Aufträge nach 1999 abgegeben haben. Der Wert für die Kundennummer in der WHERE-Klausel der Unterabfrage korreliert mit dem aus der äußeren Abfrage. Für jeden Datensatz der Tabelle kunde gibt die äußere Abfrage einen Wert an die Unterabfrage. Die wertet diese Angabe in ihrer WHERE-Bedingung aus und gibt entweder eine, mehrere oder keine Zeilen zurück. Wird eine Unterabfrage mit dem Schlüsselwort `EXISTS` begonnen, so wird die Unterabfrage in der Form ausgewertet, dass bei vorhandenen Zeilen `TRUE` und bei keinen zurückgegebenen Zeilen `FALSE` als Ergebnis der Unterabfrage steht. Die äußere Abfrage berücksichtigt den momentan auszuwertenden Datensatz aus der Tabelle *kunde* in der Ergebnismenge, wenn die Unterabfrage einen TRUE-Wert liefert. Um auf dieses Ergebnis zu kommen, könnte man für diesen Anwendungsfall auch mit einer Verknüpfung arbeiten. Verknüpfungen sollten (wenn möglich) dem Arbeiten mit Unterabfragen vorgezogen werden.

Unterabfragen in INSERT-Anweisungen fügen den Tabellen Daten zu, die aus anderen Tabellen kommen. Eine INSERT-Anweisung, welche die Quelldaten aus einer SELECT-Anweisung erhält ist wesentlich effizienter als mehrere einzeilige INSERT-Anweisungen.

Beispiel 4.52:

```
INSERT service
SELECT '1', bezeichnung, kundennr, auftragnr, anzstunden, 'AH', 'AH',
'12.12.2002', '12.12.2002'
FROM temp;
```

Das angegebene Beispiel zeigt, wie Daten mittels einer INSERT- zusammen mit einer SELECT-Abfrage Datensätze in die Tabelle *service* eingefügt werden. Die Daten kommen teils aus einer temporären Tabelle teils werden sie über Konstanten vorgegeben. Wichtig ist, dass die Bedingungen für die INSERT-Anweisung berücksichtigt werden, d.h. Anzahl und Reihenfolge der Spalten muss mit denen in der Tabelle in die geschrieben werden soll übereinstimmen. Außerdem muss man sich über die *Domänen* informieren und insbesondere die Konstanten auch dementsprechend wählen. Werden Spalten ausgelassen, müssen dort in der Tabelle Standardwerte vorhanden oder NULL-Werte erlaubt sein. Ähnlich der Verwendung einer Unterabfrage bei der INSERT-Anweisung ist die Handhabung bei der UPDATE- und DELETE-Anweisung.

4.6.5 Kombinieren mehrerer Ergebnismengen

Bei einer Abfrage können Daten aus verschiedenen Tabellen mit einer Verknüpfung zusammengefasst werden. Die Ergebnismenge weist dann Spalten aus den Quelltabellen auf. Daten aus verschiedenen Tabellen können aber auch zeilenweise zusammengefasst werden. Dies ist möglich wenn zumindest zwei SELECT-Anweisungen mit dem UNION-Operator kombiniert werden. So werden die Ergebnisse von mehreren Abfragen in einer Ergebnismenge zusammengefasst.

Beispiel 4.53:

```
SELECT  kundennr, auftragnr, anzstunden, 'temp' id FROM temp
UNION
SELECT kundennr, auftragnr, anzstunden, 'service' FROM service;
```

kundennr	auftragnr	anzstunden	id
BEFH0002	6	2	temp
BEFH0002	6	5	temp
BEKL0001	6	NULL	service
BEKL0001	6	4	service
BEKL0001	6	5	service
BEKL0001	6	10	service
BEKL0001	6	23	service
BEKL0001	6	50	service

Abbildung 4.26: Verwendung des UNION-Operators

Anhand der *id* ist ersichtlich, welche Zeilen die erste, und welche die zweite Abfrage liefert. Standardmäßig werden Duplikate im Ergebnis entfernt. Wird die ALL-Option verwendet, werden jedoch alle Zeilen inklusive Duplikate angezeigt.

4.6.6 Erstellen von temporären Tabellen

Eine wertvolle Hilfe beim Erstellen von Berichten oder bei aufwändigen Algorithmen sind *temporäre Tabellen*. Die Ergebnismenge von einer oder mehreren Abfragen kann in einer temporären Tabelle, deren Platz nach Gebrauch wieder freigegeben werden kann, eingehen. Daten können aus den verschiedenen Basistabellen zusammengetragen werden und weitere Bearbeitungsschritte können dann an der temporären Tabelle einfach durchgeführt werden. Man unterscheidet *lokale* und *globale* temporäre Tabellen. Eine lokale temporäre Tabelle ist nur in der aktuellen Sitzung sichtbar und der Platz wird nach Beendigung der Sitzung wieder freigegeben. Der Platz für eine globale temporäre Tabelle wird erst dann wieder zur Verfügung gestellt, wenn die Tabelle in keiner Sitzung verwendet wird. Globale Tabellen sind in allen Sitzungen sichtbar.

Außerdem gibt es deklarierte temporäre Tabellen, auf die man nur über Prozeduren mit einem entsprechenden auswertenden Modul zugreifen kann. Dieser Typ unterschiedet sich auch in der Art der Erzeugung von den anderen beiden. Es soll hier nur auf die erzeugten lokalen und globalen temporären Tabellen näher eingegangen werden.

Syntax:

```
CREATE {LOCAL | GLOBAL} TEMPORARY TABLE basistabelle
( basistabelle-element-kommaliste)
[ ON COMMIT {PRESERVE  |  DELETE} ROWS ]
```

Beschreibung:

CREATE {LOCAL | GLOBAL} TEMPORARY TABLE
Schlüsselbezeichnung für die Erzeugung einer lokalen oder globalen temporären Tabelle.

basistabelle
Basistabelle(n), welche die Datenquelle bildet.

basistabelle-element-kommaliste
Spalten, die aus der Basistabelle kommen und die Bestandteil der temporären Tabelle sein sollen.

ON COMMIT {PRESERVE | DELETE} ROWS
Diese Klausel definiert, ob die Tabelle bei jedem COMMIT automatisch geleert (DELETE) werden soll oder unverändert bleibt (PRESERVE)

Beispiel 4.54:

```
CREATE TEMPORARY TABLE temptable (
     idINTEGER,
     nameVARCHAR(30),
     eigenschaftVARCHAR(30));
```

4.7 Spracherweiterung durch SQL-99

Im Jahr 1999 wurde die neue Norm unter dem Namen SQL-99 veröffentlicht. Sie ist sehr umfangreich und die Spezifikationen sind auf einigen tausend Seiten festgehalten. Der neue Standard ist weitestgehend kompatibel zu SQL-92. Die Einfachheit und Übersichtlichkeit des alten Standards und damit auch des relationalen Modells geht mit dem neuen Standard, der sehr viele objektorientierte Konzepte einbindet, verloren. Die Datenstrukturen werden komplexer und die Regeln zu ihrer Anwendung werden komplexer. Das wird Konsequenzen für die weitere Nutzung von Datenbanken haben. Eine wesentliche Konsequenz wird sein, dass nicht mehr jeder normale Datenbank-

anwender verstehen kann, wie der Datenbestand organisiert ist. Und der Designer einer Datenbank hat mit viel komplexeren Entwurfsentscheidungen zu kämpfen. Die wesentlichen Änderungen beziehen sich auf die folgenden Bereiche:

- Objektorientierung,
- abstrakte Datentypen,
- Trigger,
- Multimedia,
- gespeicherte Prozeduren und Funktionen.

4.7.1 Objektorientierung

Dem Umstand, der stetig wachsenden Verbreitung von objektorientierten Systemen, wurde man durch Einbindung von objektorientierten Konzepten in den neuen Standard gerecht. Die Konzepte der *komplexen Objekte*, *Objektidentität*, *Klassen* und *Vererbung*, die aus dem Objektorientierten Modell bekannt sind, können teilweise mit dem neuen Standard umgesetzt werden.

Die Objektidentität, die einem Objekt über seine Lebenszeit hinweg eine eindeutige und unveränderbare Identität zuweist, kann beispielsweise über einen systemvergebenen Schlüssel umgesetzt werden. Jede neue Zeile in der Tabelle, die für ein neues Objekt steht, wird dann automatisch vom DBMS mit einem Schlüssel versehen.

SQL-99 unterstützt auch zwei Arten von Vererbungshierarchien: die Typhierarchie und die Tabellenhierarchie. Bei der *Typvererbung* werden ausgehend von einem Obertyp Attribute und Methoden an den Untertyp vererbt.

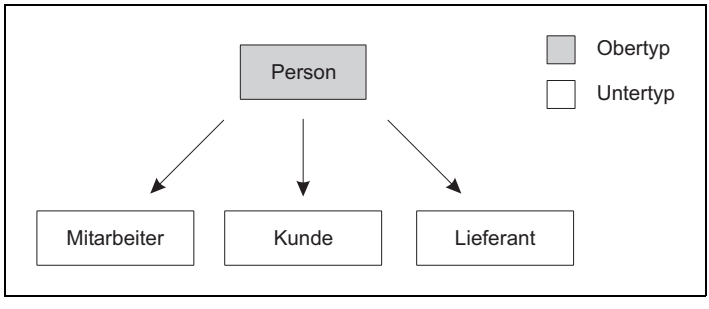


Abbildung 4.27: Typvererbung in SQL-99

Die Abbildung 4.27 zeigt einen Obertyp *Person*, der Attribute wie z.B. *ID*, *Name* und *Adresse*, an die Untertypen *Mitarbeiter*, *Kunde* und *Lieferant* vererben könnte.

4.7.2 Abstrakte Datentypen

Die vordefinierten Basisdatentypen, die wir im Abschnitt 4.2.1 kennen gelernt haben, wurden in SQL-99 erweitert (z.B. BOOLEAN und LOB). Außerdem wurde das Konzept der *abstrakten* (benutzerdefinierten) *Datentypen* implementiert. Ein Objekt kann somit komplex aufgebaut sein und der Zugriff nur über objektspezifische Methoden oder Operationen erfolgen kann. Mit der CREATE TYPE-Anweisung können abstrakte Datentypen (ADTs) mit oder ohne Objektidentität erzeugt werden.

Beispiel 4.55:

```
CREATE TYPE  kunde AS (
     KundenNrINTEGER,
     kundenNameVARCHAR(25),
     ansprechpartnerVARCHAR(25),
     ...
)
```

Das Beispiel 4.55 zeigt, wie ein benutzerdefinierter Typ, unter Verwendung der Basistypen, erstellt wird.

4.7.3 Trigger

Trigger wurden im Standard SQL-92 nicht funktional abgedeckt. Dennoch sind Trigger schon längst ein fester Bestandteil von relationalen DBMS. Besonders zur Integritätssicherung werden *Triggermechanismen* häufig verwendet. Ausgelöst durch eine Datenbankoperation wird beim Trigger ein SQL-Stapel ausgeführt. Das Konzept des Triggers wurde nun auch im Standard SQL-99 eingeführt. Auslösende Ereignisse sind DELETE-, UPDATE- und INSERT-Anweisungen, die auf Sichten oder Basistabellen ausgeführt werden. Der Ausführungszeitpunkt kann in SQL-99 vor oder nach Datenbankoperation sein. Definiert wird der Zeitpunkt durch die Angabe BEFORE oder AFTER in der CREATE TRIGGER-Anweisung. Eine ausführliche Einführung zum Arbeiten mit Triggern bietet das Kapitel 5.

4.7.4 Multimedia

Auch bei der Multimedia-Unterstützung führt der neue Standard Konzepte ein, die bei den DBMS verschiedener Hersteller bereits gängige Praxis sind. So werden z.B. LOB-Datentypen in SQL-99 eingeführt, um auf komplexe Objekte, wie Video- oder Bilddateien, zugreifen zu können. Damit diese Objekte effektiv verarbeitet werden können, werden typenspezifischen Zugriffspfade und Speicherstrukturen mit den dazugehörigen Methoden zur Verfügung gestellt. Im Kapitel 12 werden wir uns mit dieser Thematik auseinandersetzen.

4.7.5 Gespeicherte Prozeduren und Funktionen

Eine weitere Erweiterung finden wir beim Konzept der »Persistent Stored Modules« (PSM), die ebenfalls zum Standard SQL-99 gehören. In gespeicherten Modulen werden Prozeduren und Funktionen definiert, die mit einer entsprechenden Anweisung im Bedarfsfall aufgerufen werden. Dabei muss es sich nicht zwingend um SQL-Routinen handeln. Es können auch externe Routinen in C, Pascal, COBOL oder Java verwendet werden. Die Verwendung von gespeicherten Prozeduren wird in Kapitel 5 thematisiert.

4.8 Eingebettetes SQL und dynamisches SQL

Neben der Nutzung als interaktive Kommandosprache im Dialogbetrieb besteht die Möglichkeit, SQL-Anweisungen im Rahmen der üblichen problemorientierten Programmiersprachen einzusetzen. Diese Programmierspracheneinbettung wird als *Embedded SQL* bezeichnet und stellt zwei Mechanismen zur Verfügung: (Embedded) Static SQL und *Dynamic SQL*.

Beide Mechanismen können gleichzeitig in einem Programm benutzt werden. Der Effekt jeder eingebetteten SQL-Anweisung wird im Rahmen einer speziellen Variablen, genannt SQLCODE, dem Programm zugänglich gemacht.

Im Rahmen von Static SQL können SQL Anweisungen direkt innerhalb eines Programms kodiert werden. Die Datenübergabe kann dabei durch Nutzung von Konstanten oder in SQL-Anweisungen eingebettete und speziell kenntlich gemachte Programmvariablen erfolgen, die im Rahmen von SQL als Wirtsvariablen (Host Variables) bezeichnet werden. Dabei gibt es grundsätzlich keine Restriktionen bezüglich der Verwendbarkeit von SQL-Anweisungen innerhalb der Programmiersprache.

Da Programmiersprachen im Regelfall nicht mengenorientiert sondern satzorientiert sind, wird der Zugriff auf Ergebnisrelationen, für die Wirtsvariablen bereit gestellt wurden, durch einen sequentiellen Einlesemechanismus bewerkstelligt, der als *Cursor* bezeichnet wird. Durch Nutzen so genannter FETCH-Anweisungen kann in eine Wirtsvariable nacheinander jede Zeile einer Ergebnismenge geladen werden.

Alle in ein Programm eingebetteten SQL-Anweisungen, die statisch sind, werden von einem Vorübersetzer in Funktionsaufrufe einer (internen) Datenbankschnittstelle der Zielprogrammiersprache übersetzt. Zusätzlich wird ein spezieller Binder verwendet, auf den hier nicht weiter eingegangen wird. Da mit diesem Mechanismus natürlich keine Anweisungen dynamisch zur Laufzeit an eine Datenbank abgesetzt werden können, besteht zusätzlich (eingeschränkt) die Möglichkeit, dynamisch Anfragen zu formulieren.Die Formulierung von dynamischen Anfragen kann durch Programmvariablen erfolgen, die die fragliche Anweisung enthalten. Zur Ausführung der

Anweisung ist dann der Aufruf spezieller PREPARE- und EXECUTE-Anweisungen erforderlich. Die Datenübergabe von Programmvariablen erfolgt ähnlich wie beim statischen SQL. Die dynamische Konstruktion von SQL-Anfragen muss natürlich mit einem verschlechterten Laufzeitverhalten erkauft werden.

4.8.1 Einbettung in eine Wirtssprache

Was passiert nun, wenn SQL-Anweisungen in einer *Wirtssprache* wie Java oder C eingebettet werden? Die Abbildung 4.28 zeigt das grundsätzliche Prinzip für die Einbettung. Die Struktur des Datenbankprogrammes mit den SQL-Anweisungen entspricht einem üblichen Programm in der entsprechenden Programmiersprache. Für den Datenbankzugriff werden nun SQL-Befehle mit vorangestellten Kennbefehlen, wie »EXEC SQL«, in den Programmcode aufgenommen.

Die Mischung aus SQL-Anweisungen und Anweisungen der Wirtssprache werden von einem Präcompiler vorübersetzt. Dabei wird der Quellcode des gesamten Programmes nach SQL-Anweisungen durchsucht, um diese nach Korrektheit zu überprüfen und durch Unterprogramme der Wirtssprache zu ersetzen. Das reine Wirtsprogramm wird anschließend vom Compiler übersetzt. Im letzten Schritt werden durch den Linker erforderliche Module, insbesondere auch SQL-Module für das DBMS, hinzugebunden. Nach dem Prozess haben wir ein übersetztes, ablauffähiges Programm.

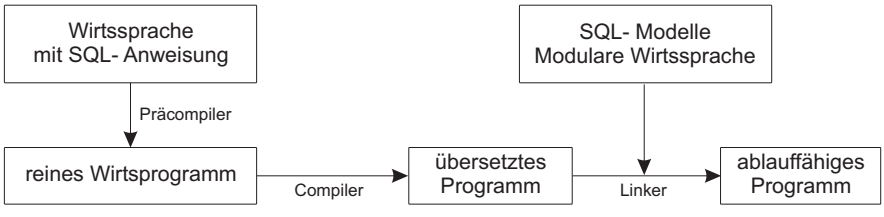


Abbildung 4.28: SQL-Einbettung in die Wirtssprache

4.9 Die SQL-Dialekte

Wie bereits eingangs erwähnt, gibt es neben dem SQL-Standard einige firmenspezifische Abweichungen. Neben den Abweichungen gibt es bei fast jedem DBMS Erweiterungen, um mit den technischen Weiterentwicklungen und Anforderungen Schritt halten zu können, müssen die Datenbankanbieter neue Funktionalität anbieten, die mit den relativ alten Standards nicht realisierbar sind. Damit bei all der Weiterentwicklung in verschiedene Richtungen die Kompatibilität nicht völlig verloren geht, bleibt der Standard als Basis auf dem auch Datenaustausch stattfinden kann. Es heißt dann

oftmals, das DBMS unterstützt *ANSI SQL* in der Eingangsstufe. Teilweise entsprechen die Erweiterungen, die die Hersteller in ihre Systeme implementieren, den Entwicklungen, die in die neuen Standards eingehen. So wird das, was sich schon längst in der Praxis bewährt hat, zum offiziellen Standard. Die Tabelle zeigt einige DBMS und die dort verwendeten Dialekte.

DB-Hersteller	DBMS	Datenbanksprache
Microsoft	SQL Server/Access	TSQL
Oracle	Oracle	PL/SQL
IBM	DB2	SQL/DS
Postgres	PostgreSQL	Postgres

Tabelle 4.9: Dialekte der Datenbanksprache

In diesem Abschnitt soll jedoch nur über die Besonderheiten von TSQL und PostgreSQL berichtet werden.

4.9.1 Datentypen

Wenn Sie die Dokumentation Ihres DBMS zur Hand nehmen, werden Sie feststellen, dass es neben den Basisdatentypen, die auch im Standard-SQL definiert sind, weitere Datentypen gibt. So finden wir bei PostgreSQL z.B. einige geometrische Datentypen, die weder bei ANSI SQL noch bei TSQL zu finden sind.

ANSI-SQL	TSQL	PostgreSQL
BIT	BIT	bit
BIT VARYING	-	BIT VARYING(*n*)
boolean	-	boolean
char	CHAR	CHARACTER(*n*)
varchar	VARCHAR	CHARACTER VARYING(*n*)
date	DATETIME	date
double precision	double precision	double precision
integer	INTEGER	integer
interval	-	interval
numeric	NUMERIC	numeric
decimal	DECIMAL	NUMERIC(*p*, *s*)
real	REAL	real
SMALLINT	SMALLINT	smallint

Tabelle 4.10: Abweichen vom SQL-Standard am Beispiel Datentypen

ANSI-SQL	TSQL	PostgreSQL
time	DATETIME	time
timestamp	ROWVERSION	timestamp
-	NCHAR(n)	-
-	NVCHAR(n)	-
-	TEXT	text
-	FLOAT	-
-	BIGINT	bigint
-	TINYINT	-
-	MONEY	money
-	SMALLMONEY	-
-	SMALLDATETIME	-
-	BINARY	-
-	IMAGE	-
-	CURSOR	-
-	TABLE	-
-	UNIQUEIDENTIFIER	-
-	-	inet
-	-	line
-	-	cidr
-	-	polygon

Tabelle 4.10: Abweichen vom SQL-Standard am Beispiel Datentypen (Forts.)

Damit Sie Ihre Daten aus der Datenbank auch ohne größere Probleme mit Fremdsoftware weiterverarbeiten wollen, sollten Sie darauf achten, dass Sie auf die Standarddatentypen zurückgreifen. Außerdem reduziert sich dadurch der Aufwand, wenn Sie mit Ihrer Anwendung ein anderes DBMS abfragen.

4.9.2 Datum- und Zeitfunktionen

Besonders bei betriebswirtschaftlichen Abläufen, wie es auch bei der Beispieldatenbank der Fall ist, spielen Zeit- und Datumsangaben eine wichtige Rolle. Leider gibt es gerade bei den Datentypen und Funktionen für diesen Bereich einige Differenzen bzgl. der Syntax und den angebotenen Funktionen zwischen den einzelnen DBMS.

Möchten Sie beispielsweise die Tagesangabe aus einem Datum extrahieren, so würde die Anweisung bei PostgreSQL wie folgt definiert:

Beispiel 4.55:

```
SELECT EXTRACT(DAY FROM TIMESTAMP '2002-02-22 20:28:40');
```

Das gleiche Ergebnis erzielen Sie mit der TSQL-Anweisung:

```
SELECT DAY('2002-02-22 20:28:40');
```

Auch an dieser Stelle kann beobachtet werden, dass die Syntax von PostgreSQL näher am Standard ist, als die von TSQL. Wollen Sie das aktuelle Datum und die aktuelle Uhrzeit ermitteln, ist dies in PostgreSQL mit gleich drei Anweisungen möglich.

Beispiel 4.56:

```
SELECT CURRENT_TIMESTAMP;
SELECT now();
SELECT TIMESTAMP 'now';
```

Arbeiten Sie mit TSQL, so verwenden Sie eine von den beiden Funktionen:

```
SELECT CURRENT_TIMESTAMP;
SELECT GETDATE();
```

Wenn Sie die Auswahl haben, so sollten Sie mit den Anweisungen arbeiten, die dem ANSI-Standard entsprechen. Sind Sie sich nicht sicher, mit welchem Datentyp Sie bei den Datumsangaben arbeiten sollen, da Sie noch nicht wissen, ob das Statistik- oder Auswertungsprogramm mit dem Sie Daten nachbearbeiten dieses Format auch unterstützt, verwenden Sie einen Zeichenfolgendatentyp. Das Arbeiten mit Zeichenfolgendatentypen bei Datums- und Zeitangaben gibt Ihnen ein Höchstmaß an Flexibilität, erfordert aber zusätzliche Mechanismen für die Integritätssicherheit.

4.9.3 Regeln

Regeln werden verwendet, um Werte in einer Spalte zu beschränken. CHECK-Einschränkungen sind darüber hinaus präziser als Regeln; auf eine Spalte kann immer nur eine *Regel* angewendet werden. Im Standard SQL-92 ist dieses Verfahren nicht berücksichtigt. Beim SQL-Server ist dieses Objekt nur noch aus Gründen der *Abwärtskompatibilität* vorhanden und wird funktional vollständig durch eine *CHECK-Anweisung* ersetzt. PostgreSQL bietet auch ein Regel-System, allerdings mit einer abweichenden Funktionalität. Hier hat der Entwickler die Möglichkeit, alternative Aktionen für INSERT-, DELETE- oder UPDATE-Anweisungen zu definieren, die anstatt einer vordefinierten Anweisung ausgeführt werden.

4.9.4 Rollen und Gruppen

Rollen und *Gruppen* spielen bei der Vergabe von Berechtigungen für Benutzer eine wichtige Rolle. Beim SQL-Server können Sie mehrere Benutzer in einer Einheit, in einer so genannten Rolle, zusammenfassen. Bei PostgreSQL ist das Äquivalent zur *Rolle* eine Gruppe. In SQL-92 gibt es keine Rollen und Gruppen. Weiteres zur Definition und Anwendung dieser Datenbankobjekt erfahren Sie im Kapitel 9.

4.9.5 Systemfunktionen

Neben den in Abschnitt 4.2.4 beschriebenen Funktionen, hat jedes DBMS weitere Systemfunktionen. Systemfunktionen liefern Werte aus Systemtabellen oder Systemdateien. Mit der Funktion `CURRENT_USER` können sowohl in PostgreSQL, als auch beim SQL Server den Benutzer der aktuellen Sitzung herausfinden. Auch hier gibt es jedoch wieder eine Vielzahl von Funktionen, die herstellerspezifisch sind. Bevor Sie mit Systemfunktionen arbeiten, informieren Sie sich in der Dokumentation Ihres DBMS. Eine Übersicht der wichtigsten Funktionen finden Sie auch im Anhang B dieses Buches.

5 Datenbank erstellen

Ist der Datenbankentwurf fertig, so ist die darauf folgende Aufgabe die Erstellung der Datenbank. Hierfür sind die SQL-Kenntnisse, die in den vorherigen Kapiteln vermittelt wurden, erforderlich. Bei einigen DBMS ist es möglich, die einzelnen Komponenten mittels grafischer Oberfläche und Assistenten zu erzeugen und konfigurieren. Die Alternative hierzu ist das Anwenden der SQL-Anweisungen in einem Skript, welches bei der Ausführung sämtliche Datenbankkomponenten erzeugt. Nicht alle hier beschriebenen SQL-Anweisungen gehören zum Standard. Bei einigen Befehlen wird deshalb die Syntax vom MS SQL Server und PostgreSQL verwendet.

5.1 Datenbankeigenschaften

Unter Datenbankeigenschaften fällt im weitesten Sinne eigentlich alles, was sie physisch und logisch ausmacht. Die Architektur der Datenbanksysteme wurde bereits vorgestellt, die Struktur der Abfragesprache SQL wurde ebenfalls gezeigt und die den DBMS zugrundeliegenden Datenmodelle sind bereits bekannt. Einige der Eigenschaften sind vom Hersteller vorgegeben und man hat weder als Anwender noch als Programmierer die Möglichkeit, daran etwas zu ändern. Vielleicht fehlt ihnen beim Arbeiten mit der Abfragesprache SQL eine Anweisung, die ihnen aus anderen Quellen bekannt ist und sie müssen feststellen, dass das von ihnen ausgewählte System diese Anweisung nicht unterstützt. In diesem Abschnitt soll es insbesondere um Eigenschaften gehen, die entweder bei der Erstellung der Datenbank oder auch nachträglich durch Konfiguration des Systems vom Programmierer oder Administrator beeinflusst werden können.

In MS Access können die Datenbankeigenschaften über DATEI und DATENBANK-EIGENSCHAFTEN in einem Fenster zur Anzeige gebracht werden (siehe Abbildung 5.1).

Zunächst erhält man hier die Informationen über Name und Pfad der Datenbankdatei sowie Größe und Erstellungsdatum. Ein eventueller Schreibschutz wird ebenso angezeigt wie eine spezielle Form der Datenbank (Archiv, System oder Verborgen). In einer weiteren Ansicht können sämtliche Datenbankobjekte, die zu dieser Datenbank gehören, aufgelistet werden. Dies ist in Abbildung 5.2 zu sehen.

Abbildung 5.1: Eigenschaften in MS Access

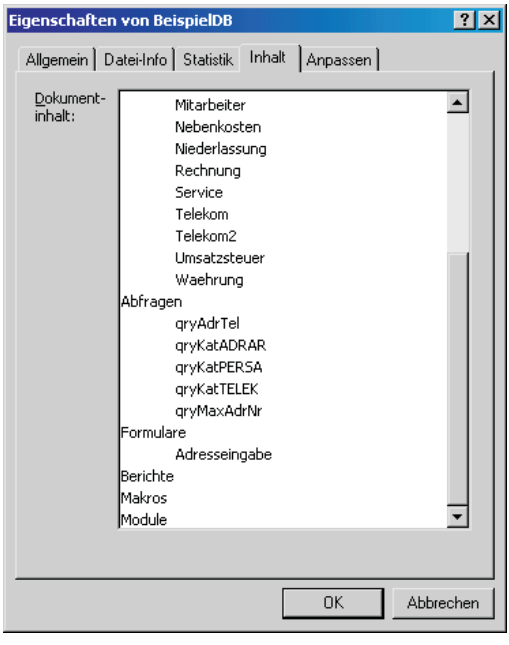


Abbildung 5.2: Datenbankobjekte in MS Access

Bei MS Access ist die Anzahl von Objekttypen überschaubar (Tabellen, Abfragen, Formulare, Berichte, Makros und Module). Die einzelnen Objekte haben auch wieder spezifische Eigenschaften, die man sich anzeigen lassen und verändern kann. Diese Eigenschaften werden beschrieben, wenn es darum geht, sie als Objekt in der Datenbank zu implementieren.

Beim MS SQL Server lassen sich weitere Informationen im Dialogfeld EIGENSCHAFTEN (Properties) anzeigen und Optionen festlegen.

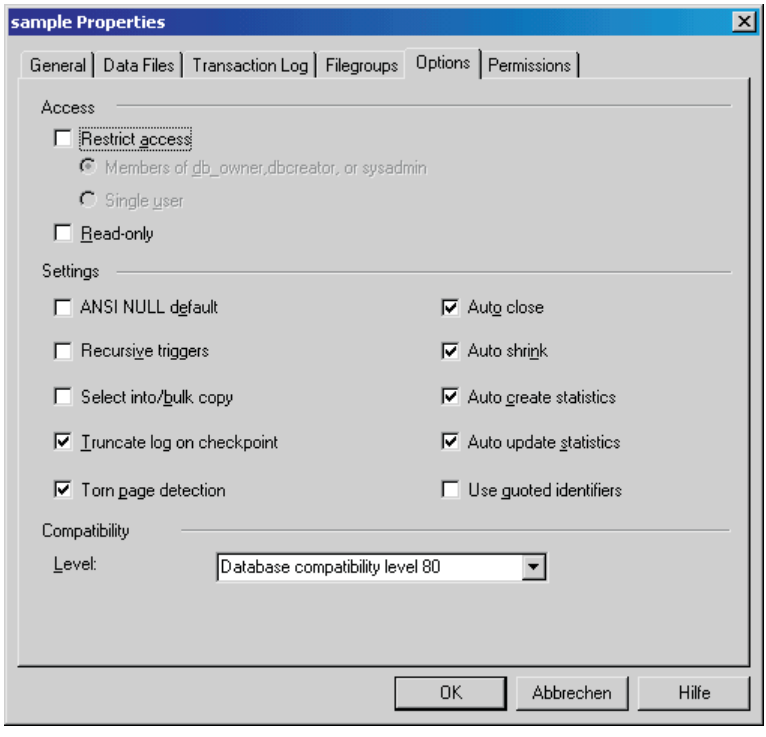


Abbildung 5.3: Eigenschaften beim MS SQL Server

In der Abbildung 5.3 ist die Registerkarte OPTIONEN ausgewählt. Hier lassen sich die Standardeinstellungen für den Datenzugriff, Statistiken, Trigger, Kompatibilitätstufe und weiteren Mechanismen einsehen und ggf. verändern. Sehr interessant zu wissen ist z.B., ob NULL-Werte zugelassen werden, wenn dies nicht bei der Definition der Spalte über eine SQL-Anweisung angegeben wurde. Für alle benutzerdefinierten Datentypen oder Spalten, die beim Erstellen oder Ändern einer Tabelle nicht explizit als NOT NULL definiert sind, gilt als Standardeinstellung das Zulassen von NULL-Werten. Beim MS SQL Server ist die Voreinstellung NOT NULL. Das DBMS kann

außerdem angewiesen werden, über AUTO CLOSE die Datenbank automatisch zu schließen, sobald sich alle Benutzer abgemeldet haben und kein Prozess aktuell ausgeführt wird.

Man kann über den COMPATIBILITY LEVEL der aktuellen Datenbank eine Verhaltensweise aufzwingen, die einer vorherigen Version entsprechen würde. Im Allgemeinen ist Abwärtskompatibilität ein Begriff, mit dem man in der Werbeabteilung versucht Pluspunkte zu machen, der im praktischen Detail jedoch nicht immer funktioniert. In den weiteren Registerkarten des Dialogfeldes EIGENSCHAFTEN (PROPERTIES) kann die Größeneistellung für Protokoll- und Datendatei erfolgen, Dateigruppen spezifiziert werden und Berechtigungen für die verschiedenen Datenbankobjekte vergeben werden.

5.2 Datenintegrität oder Korrektheit der Daten

Datenintegrität wurde bereits erwähnt und es wurde auch auf ihre Wichtigkeit hingewiesen. Doch so richtig defininiert wurde der Begriff bisher nicht. Die Datenintegrität bezieht sich auf die *Konsistenz* und Genauigkeit der Daten, die in einer Datenbank gespeichert werden. Konsistenz wird hier im Sinne von »logischer Richtigkeit« verwendet. Um dies zu veranschaulichen, soll ein Beispiel der *Inkonsistenz* angeführt werden.

Beispiel 5.1:

Wir nehmen an, dass das Bundesamt für Wehrdienst in einer Datenbank alle männlichen Bundesbürger erfaßt hat. In dieser Datenbank gibt es eine Tabelle, welche die Besoldung der aktiven Soldaten und den bereits ausgezahlten Sold kummuliert. Außerdem wird in einer Spalte angegeben, ob der Bundesbürger einen positiv entschiedenen Verweigerungsantrag gestellt hat oder nicht (dabei kann es sich um ein JA/NEIN-Feld handeln). Es darf also nur zu einer Soldauszahlung kommen, wenn beim »positiv entschiedenen Verweigerungsantrag« ein »NEIN« steht. Ist nun in dieser »Verweigerungs«-Spalte ein »JA« gesetzt und gleichzeitig sind für diesen Bundesbürger Datensätze vorhanden, die eine Soldauszahlung anzeigen, so liegt offensichtlich ein Dateninkonsistenz vor. Anders ausgedrückt, es handelt sich um einen Verstoß der Datenintegrität.

Damit solche Erscheinungen vom DBMS erkannt und im Idealfall auch bereinigt werden können, müssen Integritätsregeln bestimmt und Mechanismen in Kraft gesetzt werden, welche diese überwachen. In dem geschilderten Fallbeispiel würde der Fehler wahrscheinlich erst erkannt, wenn sich die betreffende Person beschweren (was bei einer Auszahlung fraglich ist) oder ein Plausibilitätscheck über die Datenbank laufen würde. Integritätsverletzungen können jedoch mit entsprechenden Vorkehrungen wesentlich früher, oftmals bereits bei der Dateneingabe, festgestellt werden.

Die verschiedenen Typen der Datenintegrität sehen wir in Abbildung 5.4.

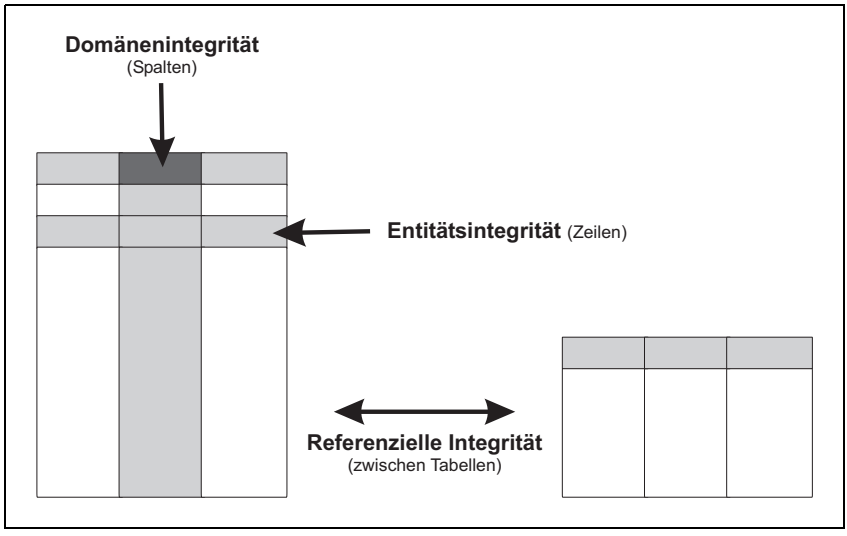


Abbildung 5.4: Typen der Datenintegrität

Geht es darum, dass die Werte einer Spalte korrekt sind, d.h. der vorgegebene Wertebereich (Domäne) für diese Spalte eingehalten wird, so wird dies *Domänenintegrität* genannt. Die Domänenintegrität wird oft durch die Verwendung der Gültigkeitsprüfung erzwungen. Sie kann jedoch auch durch Einschränkung der Datentypen, das Format oder den zulässigen Wertebereich in einer Spalte erzwungen werden. In einer Spalte soll zum Beispiel ausgeschlossen werden, dass NULL-Werte zulässig sind. Das kann mit einer Einschränkung, die den angegebenen Wert überprüft, verhindert werden.

Bei der *Entitätsintegrität* wird die Zeile (oder der Datensatz) einer Tabelle betrachtet. Alle Zeilen einer Tabelle benötigen einen eindeutigen Bezeichner, mit dem dieser Datensatz und damit das Objekt (Entität), welches hinter diesem Datensatz steht, selektiert werden kann. Jede Zeile einer Tabelle sollte eindeutig identifizierbar sein, dann ist die Entitätsintegrität gewährleistet. Bei Verwendung von eindeutigen Indizes hat man automatisch einen eindeutigen Bezeichner für jede Zeile.

Die *referenzielle Integrität* garantiert die Beibehaltung der Beziehung zwischen Tabellen, welche über Primär- und Fremdschlüssel aufgebaut wurde. Die Referenzen einer Tabelle sind auch bzgl. der Richtigkeit von Beziehungsarten zu überprüfen. Eine Zeile in einer referenzierten Tabelle kann nicht gelöscht werden oder der Primärschlüssel geändert werden, wenn ein Fremdschlüssel auf diese Zeile verweist. Die praktische Bedeutung soll auch hier wieder ein Fallbeispiel verdeutlichen.

Beispiel 5.2:

In einem Wohltätigkeitsverein werden die monatlichen Beiträge der Mitglieder per Lastschriftverfahren eingezogen. Sämtliche Mitglieder sind in einer Datenbank erfasst und befinden sich in der Tabelle *mitglieder*. Der monatlich abzubuchende Beitrag steht in einer Tabelle *mitgliederbeitrag*. Ein Mitglied ist verstorben und der zugehörige Datensatz wurde korrekt in der Tabelle *mitglieder* gelöscht. In der Tabelle *mitgliederbeitrag* steht jedoch nach wie vor der Datensatz für das verstorbene Mitglied. Das Konto wurde mittlerweile aufgelöst und das System erzeugt nun automatisiert ein Mahnungsanschreiben an das Mitglied.

Die Angehörigen werden sich vielleicht über die Raffgier des Vereins wundern. Ein kleiner Fehler. Eine schlecht (oder gar nicht) implementierte Integritätsüberprüfung, kann somit zu einem Imageschaden führen. Ein guter Entwurf und ein richtig konfiguriertes System, würde eine »Datensatzleiche«, wie im angegebenen Beispiel gar nicht zulassen. Beim Löschen des Datensatzes in der Tabelle *mitglieder* müssten die referenziellen Tabellen, hier die Tabelle *mitgliederbeitrag*, ebenfalls überprüft werden. Eine Meldung an den Benutzer oder ein automatisches Löschen des Datensatzes der Tabelle *mitgliederbeitrag*, hätten dem Verein die Peinlichkeit erspart.

Es gibt zwei Methoden, um die Datenintegrität zu erzwingen: *deklarativ* und *prozedural*. Bei der *deklarativen Datenintegrität* werden Kriterien definiert, welche die Daten als Teil einer Objektdefinition erfüllen müssen. Das jeweilige DBMS stellt dann sicher, dass diese Kriterien auch erfüllt werden. Bei der *prozeduralen Integrität* werden SQL-Skripte erstellt, die sowohl die Kriterien festlegen, die die Daten erfüllen müssen, als auch diese Kriterien erzwingen. Die bevorzugte Methode, um grundlegende Datenintegrität zu implementieren, ist die Verwendung der deklarativen Integrität. Dir prozedurale Integrität sollte auf kompliziertere Geschäftsabläufe oder andere Ausnahmen beschränkt bleiben.

Integritätstyp	**Methode**	**Art der Methode**
Domäne	Default	Einschränkung – deklarativ
	Check	Einschränkung – deklarativ
Entität	Primärschlüssel	Einschränkung – deklarativ
	Unique	Einschränkung – deklarativ
Referenziell	Fremdschlüssel	Einschränkung – deklarativ
	Check	Einschränkung – deklarativ
	Trigger	prozedural
	Gesp. Prozedur	prozedural

Tabelle 5.1: Zuordnung der Integritätsmechanismen

Zu den deklarativen Methoden gehört die Einschränkung (Constraints). Bei der Einschränkung wiederum werden verschiedene Typen unterschieden (siehe auch Tabelle 5.1). Wenn Einschränkungen eingerichtet oder geändert werden, sollten die folgenden Punkte berücksichtigt werden:

1. Einschränkungen können erstellt, geändert oder gelöscht werden, ohne dass die Tabelle gelöscht bzw. neu erstellt werden muss.
2. In der Anwendung, welche die Datenbank als Basis hat, muss Programmlogik integriert werden, in der Verstöße gegen Einschränkungen ausgewertet werden.
3. Das DBMS überprüft die vorhandenen Daten, wenn einer Tabelle nachträglich Einschränkungen hinzugefügt wurden.
4. Der Einschränkung sollte ein einfacher sinnvoller Name zugewiesen werden.

5.3 Erstellen der Datenbank

Wie Zirkusartisten haben wir in den letzten beiden Kapiteln mit Datenbankobjekten jongliert, ohne einen festen Boden unter den Füßen zu haben. Es wurden Tabellen erzeugt und Abfragen auf ihnen ausgeführt, ohne das Wissen wie die Datenbank selbst zu erstellen ist. Das wird in diesem Abschnitt nachgeholt. Die Kenntnisse zur Entwurfstheorie, zur Datenbankarchitektur und die Grundkenntnisse von SQL werden dabei von großem Nutzen sein.

5.3.1 Vom Entwurf zur Implementierung

Wird die Datenbank erstellt, so müssen zumindest zwei Dinge bekannt sein:

1. Wie sieht die Datenbankarchitektur aus?
2. Welcher Datenbankentwurf liegt der Datenbank zugrunde?

Die Entwurfsphase ist abgeschlossen und ausgehend von den Spezifikationen wird die Datenbank und ihre Objekte nun implementiert.

5.3.2 Datenbank erstellen

Datenbanken können in vielen DBMS mit Hilfswerkzeugen auf grafischem Weg erstellt werden. Die einzelnen Parameter werden dann von einem Assistenten abgefragt. In diesem Abschnitt soll die Erstellung der Datenbank mit der CREATE DATABASE-Anweisung im Mittelpunkt stehen. Diese Anweisung hat den Vorteil, dass sie, wenn auch in leicht abgewandelter Form (siehe DB-Dialekte), für alle RDBMS brauchbar ist und in einem Datenbankskript angewendet werden kann. Welche Vorteile ein Datenbankskript mit sich bringt, wird in dem Kapitel »Skripterstellung« beschrieben.

Syntax für MS SQL Server:

```
CREATE DATABASE database_name
[ ON [ PRIMARY ]
 ( [ NAME = logical_file_name , ]
    FILENAME = 'os_file_name'
    [ , SIZE = size ]
    [ , MAXSIZE = { max_size | UNLIMITED } ]
    [ , FILEGROWTH = growth_increment ] ) [ ,...n ]]
        [ , < filegroup > [ ,...n ] ]
]
[ LOG ON { ( [ NAME = logical_file_name , ]
    FILENAME = 'os_file_name'
    [ , SIZE = size ]
    [ , MAXSIZE = { max_size | UNLIMITED } ]
    [ , FILEGROWTH = growth_increment ] ) [ ,...n ] } ]
```

Beschreibung:

Database_name
Name der neuen Datenbank nach den Regeln für Bezeichner.

PRIMARY
Schlüsselwort, welches darauf hinweist, dass die Dateiliste die Primärdatei spezifiziert.

logical_file_name
Logischer Dateiname für die Datendatei innerhalb der SQL-Anweisungen. Der logische Dateiname kann, muss aber nicht mit dem physischen Dateinamen übereinstimmen.

os_file_name
Physischer Dateiname inklusive der Pfadangabe.

Size
Anfangsgröße der Datendatei.

max_size
Maximalgröße, auf die die Datendatei vergrößert werden darf.

UNLIMITED
Datei kann so lang vergrößert werden, bis der Speicherplatz erschöpft ist.

growth_increment
Zusätzlicher Speicherplatz, den die Datendatei bei der Vergrößerung erhält.

Filegroup
Definierte Datendateien für die primäre Dateigruppe.

LOG ON
Schlüsselwort, welches die Angaben für die Protokolldatei einleitet.

Mit Systemprozeduren und weiteren SQL-Anweisungen lassen sich zusätzliche Datenbankoptionen festlegen. Dazu gehören insbesondere Angaben zur Datensicherheit, zur Protokollierung und zur automatischen Speicherverwaltung. Weitere Datenbankobjekte wie Tabellen, Indizes, Einschränkungen etc. müssen explizit erstellt werden.

Syntax für PostgreSQL:

```
CREATE DATABASE name
       [ WITH [ LOCATION = 'dbpath' ]
           [ TEMPLATE = template ]
           [ ENCODING = encoding ] ]
```

Beschreibung:

name
Name der Datenbank.

dbpath
Datenbankpfad als Zeichenkette, in der die neue Datenbank gespeichert wird.

template
Name der Strukturschablone (Template), welche für die Erstellung der neuen Datenbank verwendet wird.

encoding
Kodiermethode, die in der neuen Datenbank verwendet wird.

Das folgende Beispiel benutzt die Syntax des MS SQL Servers, um eine neue Datenbank zu implementieren.

Beispiel 5.3:

```
CREATE DATABASE sample
  ON
   (NAME=sample,
    FILENAME = 'C:\MSSQL\Data\sample.mdf',
    SIZE =20 MB,
    MAXSIZE=50 MB,
    FILEGROWTH=10%)
   LOG ON
   (NAME = sample_log,
    FILENAME = 'D:\MSSQL\Data\sample_log.ldf',
    SIZE =5MB,
    MAXSIZE=10 MB,
    FILEGROWTH=10%)
```

In Beispiel 5.3 wird die Datenbank *sample* erstellt. Es wird eine Datendatei mit 20 Mbyte und eine Protokolldatei mit 5 Mbyte angelegt. Für beide Dateien gibt es weitere Angaben zum Pfad, maximale Dateigröße, Schrittweite und logischer Name. Die Angaben können auch nachträglich noch geändert werden.

Dateigruppen

Dateigruppen sind eine bestimmte Anzahl von Dateien, die für Reservierung- und Verwaltungszwecke als Einheit verstanden werden kann. Falls die Hardware mehrere Festplatten umfasst, können bestimmte Objekte und Dateien auf unterschiedlichen Platten abgelegt sein und gleichzeitig zu einer Datenbank mit mehreren Dateigruppen gehören. Eine Standarddateigruppe wird erzeugt, wenn die Datenbank angelegt wird. Weitere Dateigruppen können individuell erstellt werden. Protokolldateien sind nicht Bestandteil von Dateigruppen. Datenbankobjekte können in Dateigruppen zusammengefasst werden, um logische Einheiten zu bilden, die beim Wiederherstellen oder komplexen Abfragen Vorteile bieten.

5.3.3 Eigenschaften verändern

Auch bei einem vorzüglichen Datenbankentwurf und vorausschauenden Überlegungen zur erwarteten Datenmenge wird häufig eine Änderung an der Datenbank notwendig sein. Wenn sich die Datenbank vergrößert oder ändert, kann sie automatisch oder manuell, menügesteuert oder auf Befehlszeilenebene, vergrößert oder verkleinert werden. Nachträglich können auch weitere Protokoll- und Datendateien zugefügt werden.

Syntax für MS SQL Server:

```
ALTER DATABASE database
  {ADD FILE < filespec > [,...n ]
   [TO FILEGROUP filegroup_name ]
  | ADD LOG FILE < filespec > [,...n ]
  | REMOVE FILE logical_file_name [ WITH DELETE ]
  | ADD FILEGROUP filegroup_name
  | REMOVE FILEGROUP filegroup_name
  | MODIFY FILE < filespec >
  | MODIFY NAME = new_dbname
  | MODIFY FILEGROUP filegroup_name {filegroup_property
    | NAME = new_filegroup_name }
  | SET < optionspec > [,...n ] [ WITH < termination > ]
}
```

Beschreibung:

ADD FILE
gibt an, dass eine neue Datendatei hinzugefügt wird.

TO FILEGROUP
Angabe zur Dateigruppe, zu der die Datei gehören soll.

ADD LOG FILE
Es wird eine neue Protokolldatei hinzugefügt.

REMOVE FILE
Eine Datei wird logisch und physikalisch aus dem DBMS entfernt.

ADD FILEGROUP
Eine Dateigruppe wird hinzugefügt.

REMOVE FILEGROUP
Eine Dateigruppe wird aus dem DBMS entfernt.

MODIFY FILE
Eine bestehende Datenbankdatei wird verändert.

MODIFY NAME
Der Datenbankname wird verändert.

MODIFY FILEGROUP
Eine bestehende Dateigruppe wird verändert.

Weitere Angaben sind entsprechend dem CREATE DATABASE-Befehl zu verwenden. Eine nicht mehr benötigte Datenbank kann mit der Anweisung DROP DATABASE aus dem DBMS entfernt werden. In einigen Fällen ist das Ändern oder Löschen der Datenbank nicht möglich bzw. nicht empfehlenswert:

- Die Datenbank wird gerade wiederhergestellt.
- Ein Replikationsvorgang läuft gerade auf dieser Datenbank.
- Es gibt bestehende Abhängigkeiten zu Anwendungen oder anderen Datenbanken.
- Die Datenbank wird von einem beliebigen Benutzer gerade für einen Schreib- oder Lesevorgang verwendet.

Einer Datenbank soll eine sekundäre Datenbankdatei hinzugefügt werden. Diese kann physikalisch an einem anderen Ort wie die primäre Datendatei abgelegt sein. Das folgende Beispiel zeigt, wie dies mit der ALTER DATABASE-Anwendung passieren kann:

Beispiel 5.4:

```
ALTER DATABASE sample
   ADD FILE
   (NAME=sample2,
    FILENAME = 'E:\Temp\sample2.ndf',
```

```
    SIZE =10 MB,
    MAXSIZE=50 MB,
    FILEGROWTH=10%)
```

Die Beispieldatenbank umfasst auch einige Multimediaobjekte, welche in einer separaten Dateigruppe organisiert sein sollen.

Beispiel 5.5:

```
ALTER DATABASE sample
     ADD FILEGROUP service
```

Das Beispiel 5.5 hat für die Beispieldatenbank eine benutzerdefinierte Dateigruppe mit dem Namen *service* angelegt. In dieser sollen das Serviceangebot des Unternehmens organisiert werden. Grafiken und Videoclips, die im Internet abrufbar sind, werden hier abgelegt. Eine Tabelle, die neu angelegt werden soll, kann direkt dieser benutzerdefinierten Dateigruppe zugeordnet werden. Eine bereits existierende Tabelle kann nachträglich über ihre Eigenschaften der Dateigruppe zugefügt werden:

1. Die Tabelle im Enterprise Manager im Designmodus öffnen.
2. Eine Spalte mit der rechten Maustaste auswählen.
3. Eigenschaften anklicken.
4. Im Eigenschaftsfenster die Dateigruppe auswählen (siehe Abbildung 5.5).

Die Dateigruppe *service* kann nun bei besonderen Datenbankaktionen wie Replikation oder Datenbanksicherung ausgespart werden.

5.4 Indizes bringen Abfragebeschleunigung

Eine der Hauptaufgaben einer Datenbankanwendung ist das Suchen und Finden von Daten, egal ob Sie eine spezielle Kundenadresse oder die Akte für einen Patienten suchen. Selbst bei Serienbriefen oder der Berichtserstellung müssen Daten zunächst gesucht und verfügbar gemacht werden. Es muss also gesucht werden. Aber wie und wo? Wie würden Sie vorgehen, wenn Sie ein dickes Buch in der Hand haben und wissen irgendwo in diesem Buch steckt die Information, Sie wissen nur nicht auf welcher Seite und in welchem Kapitel? Berufsoptimisten würden vielleicht irgendeine Seite aufschlagen und wenn es kein Treffer war, in den benachbarten Seiten ein wenig herumblättern. Die Geduldigen würden sogar am Anfang des Buche beginnen und sämtliche Seiten, bis zur gesuchten Stelle durcharbeiten. Auf diese Weise kann man auch Zeit totschlagen. Normalerweise wird man sehr schnell dazu übergehen den entsprechende Begriff im Index oder das Thema im Inhaltsverzeichnis zu suchen.

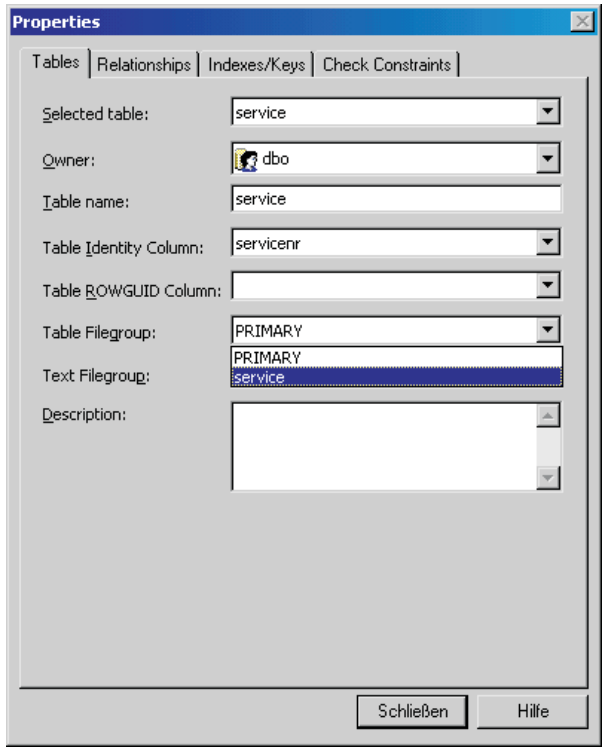


Abbildung 5.5: Erstellung einer Dateigruppe beim MS SQL Server

Entsprechend der Möglichkeiten beim Buch, gibt es auch bei der Datenbank verschiedene Methoden auf die Daten zuzugreifen. So können sämtliche Datenseiten nacheinander durchsucht werden. Dieses Verfahren nennt man *Tabellenscan*, weil alle zu einer Tabelle gehörigen Datenseiten gescannt werden. Die Zeilen, die dem Abfragekriterium entsprechen werden extrahiert. Alternativ zum Tabellenscan kann mit Hilfe von Indizes nach der Datenseite gesucht werden, ohne dass nicht relevante Datenseiten abgefragt werden. Beide Verfahren haben ihre Vor- und Nachteile, die man im konkreten Anwendungsfall gegenüber stellen muss.

5.4.1 Wie werden Daten gespeichert?

In so genannten Datenseiten werden Daten abgelegt. Datenseiten bestehen prinzipiell aus einem *Seitenkopf* (Header), welcher Angaben über die Datenseite und ihre Abhängigkeiten beinhaltet, den *Datenzeilen* und einem *Zeilenoffset*. In den Datenzeilen werden die eigentlichen Zeilen der Tabelle gespeichert. Der Speicherbereich für die Datenzeilen je Datenseite hat bei jedem DBMS eine feste Größe. Die Anzahl der auf einer bestimmten Seite gespeicherten Zeilen schwankt in Abhängigkeit von der Tabellen-

struktur und den gespeicherten Daten. Eine Tabelle, bei der die Spalten immer eine feste Länge haben, hat immer dieselbe Anzahl an Zeilen pro Datenseite. Zeilen mir variabler Länge werden in der Anzahl gespeichert, die der tatsächlichen Länge der eingegebenen Daten entspricht. Geringere Zeilenlänge ermöglicht mehr Zeilen pro Seite; das reduziert die Zugriffe und verbessert die Trefferquote im Zwischenspeicher.

Im Zeilenoffsetbereich ist Information über die logische Reihenfolge der Zeilen enthalten. Wenn eine Tabelle einen *gruppierten Index* besitzt, dann speichert das DBMS die Zeilen nach der Reihenfolge des Schlüssels für diesen gruppierten Index. Die Einteilung hat aber keine Auswirkung auf den physischen Ort der Zeile.

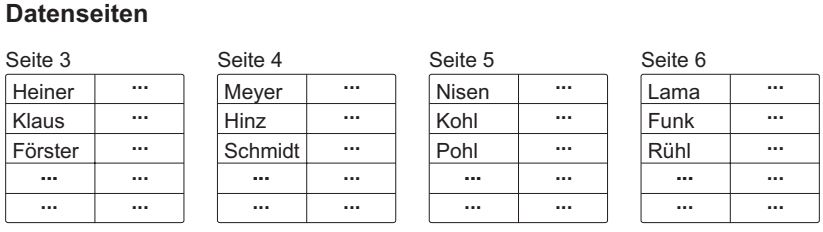


Abbildung 5.6: Aufteilung der Daten auf Datenseiten

Die Daten einer Tabelle werden auf mehrere Datenseiten fester Größe verteilt und man spricht dann häufig auch von (Daten-) *Blöcken*. Die Datenzeilen werden zunächst ohne bestimmte Reihenfolge gespeichert. Wenn einer Seite Zeilen hinzugefügt werden sollen und diese Seite voll ist, werden die Datenseiten aufgeteilt. Die Angabe über vorherige und nachfolgende Seite ist ebenfalls im Seitenkopf enthalten. In Abbildung 5.6 ist eine solche Verteilung auf Datenseiten für die Tabelle *adresse* der Beispieldatenbank grob skizziert. Es sind nur die Namen zu sehen, aber es werden dort selbstverständlich die vollständigen Datensätze gespeichert.

Der Zugriff auf Datenseiten muss schnell erfolgen. Damit der Zugriff effizient erfolgen kann wird oftmals mit einer Technik gearbeitet, die auch *Hashing* genannt wird. Verwendet werden dabei mathematische Funktionen, die Schlüsselwerte in Blocknummern umrechnen und umgekehrt. Das Ziel dabei ist eine gleichmäßige Verteilung der Sätze. Der ermittelte Schlüssel wird einer Gruppe von *Hashbehältern* zugewiesen. Ein Hashbehälter ist eine Struktur, die ein Feld von Zeigern in Form einer verknüpften Liste auf die Datenseiten enthält. Wenn nicht alle Zeiger auf eine einzige Hashseite passen, wird eine verknüpfte Liste zu weiteren Hashseiten angefügt.

5.4.2 Indextypen

Es gibt verschiedene Typen von Indizes, die sich bzgl. ihrer Struktur und der verwendeten mathematischen Algorithmen unterscheiden. Stellvertretend werden im weiteren Verlauf dieses Abschnitts die Typen vorgestellt, die beim MS SQL Server oder bei PostgreSQL auftreten können.

Eindeutiger Index

Ein eindeutiger Index stellt sicher, dass die Werte in der indizierten Spalte nur einfach vorkommen und Duplikate nicht zugelassen werden. Eindeutigkeit kann auch durch einen Index erzwungen werden, der sich auf mehrere Spalten bezieht.

Beispiel 5.6:

In einer Tabelle seien Spalten für den Namen, Geburtsdatum und Geburtsort vorgesehen. In jeder Spalte ist ein Mehrfachvorkommen der Werte zu erwarten. Namen wie »Schmidt«, »Meyer« oder »Müller« sind ebenso häufig zu erwarten, wie große Städte als Geburtsorte. Die Wahrscheinlichkeit, dass es Duplikate bei der Kombination dieser drei Spalten gibt, ist jedoch schon sehr viel geringer und kann deshalb als zusammengesetzter Index erstellt werden.

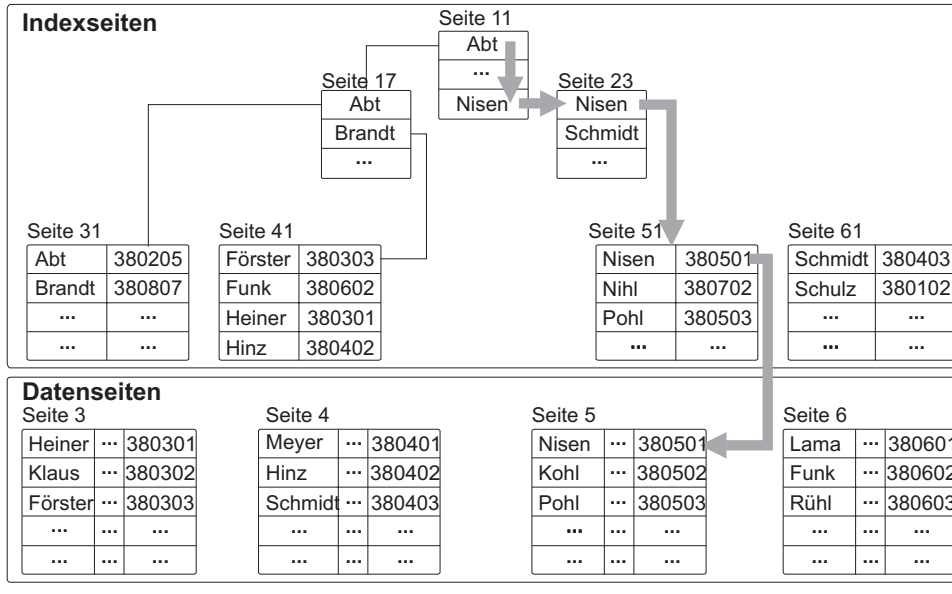


Abbildung 5.7: Datenzugriff über Indexseiten

Bei den meisten DBMS wird bei der Implementierung eines *primären Schlüssels* oder einer UNIQUE-Einschränkung automatisch ein eindeutiger Index erstellt. In dem vorgestellten Fallbeispiel ist ein primärer Schlüssel empfehlenswert, weil die Eindeutigkeit dann vom System sichergestellt wird. Ein eindeutiger Index kann auch nach Erstellung der Tabelle und sogar dann erstellt werden, wenn bereits Daten in der Tabelle vorhanden sind. Dabei ist jedoch sicherzustellen, dass die indizierten Spalten keine Duplikate enthalten.

Beim SQL Server kann ein eindeutiger Index für CLUSTERED und NONCLUSTERED Indizes verwendet werden. Bei PostgreSQL ist das nur für den Index mit dem Typ BTREE möglich.

Gruppierter Index (CLUSTERED)

Die physische Reihenfolge der Zeilen in der Tabelle ist mit der Reihenfolge der Zeilen im Index identisch. Gruppierte Indizes eignen sich für Spalten, die häufig nach Schlüsselwerten durchsucht werden. Bei der Implementierung der Indizes in einer Tabelle sollte darauf geachtet werden, dass gruppierte Indizes vor den nicht gruppierten erstellt werden, weil sich dabei die physische Reihenfolge der Tabellenzeilen ändert. Die Zeilen werden in einer *sequenziellen Reihenfolge* sortiert, die beibehalten wird. Beim gruppierten Index wird vorübergehend bei der Erstellung das 1,2-fache der Tabellengröße an Speicherplatz benötigt. Pro Tabelle kann nur jeweils ein gruppierter Index vergeben werden.

Nicht gruppierter Index (NONCLUSTERED)

Man greift auf nicht gruppierte Indizes zurück, wenn die Daten auf unterschiedliche Weise durchsucht werden. Die Reihenfolge der Seiten eines nicht gruppierten Index unterscheidet sich von der physischen Reihenfolge in der Tabelle. Dieser Indextyp wird am günstigsten für Spalten erstellt, in denen die Daten *hoch selektiv* bis eindeutig sind. Pro Tabelle können mehrere nicht gruppierte Indizes definiert werden. Die genaue Anzahl hängt vom DBMS ab. Nicht gruppierte Indizes sind der Standardindextyp beim SQL Server. Wird ein gruppierter Index erstellt oder gelöscht, so werden nicht gruppierte Indizes in dieser Tabelle neu erstellt. Die *Zeilen-IDs* bestimmen die logische Reihenfolge der Zeilen und setzen sich aus *Datei-ID*, Seitennummer und der *Zeilen-ID* zusammen.

B-Tree Index

Beim hierarchischen Datenmodell haben wir bereits Baumstrukturen kennen gelernt. B-Bäume (B-Tree) wurden in den späten 60er Jahren von *R.Bayer* und *E.McCreight* entwickelt. Bäume sind zyklenfreie, gerichtete Graphen mit einem ausgezeichneten *Knoten* (Wurzel), von dem aus man alle anderen Knoten auf genau einem Weg erreicht.

Im B-Baum befindet sich nicht mehr nur ein *Schlüssel* in einem Knoten, sondern mehrere, so dass sich der zu wählende Zweig aus mehreren Größenvergleichen je Knoten ergibt. Jeder Knoten in einem B-Baum des Ranges *d* mit Ausnahme der Wurzel enthält zwischen *d* und *2d* Schlüssel und hat somit zwischen *d+1* und *2d+1* Nachfolger. Jeder Knoten ist also mindestens zur Hälfte gefüllt. Die Wurzel enthält zwischen einem und *d* Schlüssel und hat somit mindestens zwei Nachfolger. Das Außergewöhnliche beim B-Baum, also das was ihn von anderen Bäumen unterscheidet, sind seine balancierten Äste, d.h. die Wege von der Wurzel bis zu den Blättern sind und bleiben immer gleich lang. Die Balance ist das Erfolgsrezept! Ein Ast eines binären Baumes z.B. kann beliebig lang werden, wobei sein Bruderast genau so kurz bleiben kann wie am Anfang, nicht aber bei dem B-Baum. Je balancierter der Baum ist, desto schneller erfolgt z.B. die Suche in ihm.

B-Bäume sind effizient, vielseitig und einfach. Sie lassen sich gut an die Hardware und die Bedürfnisse des Benutzers anpassen. Sie liefern gute Ergebnisse bei Zugriffen auf Sekundärspeicher und sequentiellen Zugriffen. Es ist immer eine Speicherplatzausnutzung von 50% garantiert, in manchen Varianten auch mehr. Die dynamischen Methoden der Balancierung sorgen für eine selbständige Wartung.

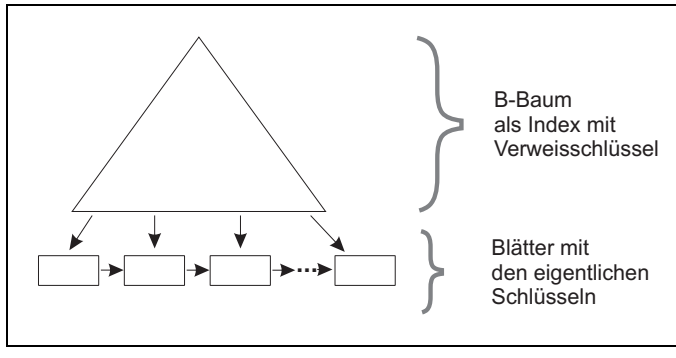


Abbildung 5.8: Schlüssel und Indizes im B-Baum

Die Abbildung 5.8 zeigt die logische Trennung zwischen Index- und Schlüsselteil. Natürlich besitzen Indexknoten und Blattknoten unterschiedliche Form und Größe. Die Blattknoten sind untereinander von links nach rechts verkettet.

R-Tree Index

Der R-Tree (R-Baum) eignet sich gut für Indizierung von Nachbarschaftsbeziehungen zwischen Objekten eines *n-dimensionalen Raumes*. Vorwiegend wird der R-Tree für 2- und 3-dimensionale Daten eingesetzt. Der R-Tree ist als Sekundärspeicherstruktur konzipiert. Das bedeutet, seine Knoten entsprechen *Sekundärspeicherseiten*, d.h. jeder Knoten wird auf einer Seite abgespeichert.

Ein R-Tree der Ordnung (n, m) besitzt in jedem Knoten mit Ausnahme der Wurzel mindestens $n <= m/2$ und höchstens m Einträge. Die Wurzel hat mindestens zwei Söhne oder gar keinen.

Die Blätter des Baumes erhalten Verweise auf konkrete Objekt in der Datenbank. Der Baum ist vollständig balanciert, d.h. alle Pfadlängen von der Wurzel zu einem Blatt sind gleich. Die Baumstruktur ist von der Einfügereihenfolge abhängig.

Hash Index

Es wurde bereits gezeigt, dass mittels *Hashfunktionen* Datenwerte in Blockwerte umgerechnet werden. Das Ergebnis dieser Umrechnung ist ein Bereich mit Zeigern, die auf die jeweiligen Datenwerte weisen. Werden nun weitere Daten hinzugefügt, muss mittels der Hashfunktion ein neuer Wert errechnet und als Zeiger abgelegt werden. Wird ein bestimmter Datenwert gesucht, wird der Datenwert erneut umgerechnet, um den Blockwert und damit den physischen Plattenplatz des gesuchten Wertes zu ermitteln. Sowohl beim Hinzufügen eines neuen Datensatzes zum Index, als auch beim Finden eines bestimmten Datensatzes werden jeweils nur zwei Zugriffe auf die Festplatte benötigt. Deshalb sind die Zugriffszeiten bei Verwendung eines Hash Index enorm kurz. Ein entscheidender Nachteil ist, dass man sich innerhalb der Datensätze nicht in einer logischen Reihenfolge »Schritt für Schritt« bewegen kann. Die Funktion »Gehe zum nächsten Datensatz« gibt es praktisch nicht.

5.4.3 Erstellen und Löschen von Indizes

Bei der Erstellung von Indizes kann die SQL-Anweisung CREATE INDEX verwendet werden. Diese Anweisung würde in der einfachsten Form einen Index für eine Spalte festlegen, wie wir das im Beispiel 5.7 sehen können.

Beispiel 5.7:

```
CREATE INDEX idx_kurzname ON adresse (kurzname);
```

Die vollständige Syntax für den **MS SQL Server** würde wie folgt aussehen:

Syntax:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
    ON { table | view } ( column [ ASC | DESC ] [ ,...n ] )
[ WITH
        [ PAD_INDEX ]
        [ [ , ] FILLFACTOR = fillfactor ]
        [ [ , ] IGNORE_DUP_KEY ]
        [ [ , ] DROP_EXISTING ]
        [ [ , ] STATISTICS_NORECOMPUTE ]
```

```
        [ [ , ] SORT_IN_TEMPDB ]
]
[ ON filegroup ]
```

Beschreibung:

CREATE
Einleitendes Schlüsselwort für die CREATE INDEX-Anweisung.

UNIQUE
gibt an, dass ein eindeutiger Index verwendet wird, d.h. in der Schlüsselspalte dürfen keine Duplikate vorkommen.

CLUSTERED | NONCLUSTERED
Bei einem CLUSTERED Index ist die physische Ordnung der Daten gleich der Zeilenordnung des CLUSTERED Index. Beim NONCLUSTERED Index ist die Reihenfolge beim Index unabhängig von der physischen Reihenfolge.

index_name
Name des Indexes. Der Name muss eindeutig innerhalb der Tabelle, jedoch nicht innerhalb der DB sein.

table | view
Name der Tabelle oder der Sicht, welche die Spalte enthalten, die indiziert werden soll.

column
Spalte oder Spaltenliste, die indiziert wird.

ASC | DESC
bestimmt die Sortierreihenfolge für die indizierte Spalte. Der Standardwert ist ASC (aufsteigend).

PAD_INDEX
In Kombination mit FILLFACTOR wird freier Platz auf einer Indexseite in Form eines Prozentsatzes festgelegt.

FILLFACTOR
Definiert, wie weit die Indexseite gefüllt werden soll.

IGNORE_DUP_KEY
Angabe über die Aktion beim Einfügen von Duplikaten. Duplikate werden beim Einfügen ignoriert und es wird eine entsprechende Meldung ausgegeben.

DROP_EXISTING
Ein bereits existierender Index des gleichen Namens wird, bei Verwendung der CREATE INDEX-Anweisung mit identischem Namen, gelöscht.

STATISTICS_NORECOMPUTE
Keine automatische, statistische Aufbereitung der Indizes, die außerhalb des Gültigkeitsdatums liegen.

SORT_IN_TEMPDB
Definition des physischen Ortes, an dem das Sortieren der Daten für die Indexerstellung vorgenommen werden soll.

ON filegroup
Der Index wird auf einer zuvor definierten Dateigruppe erzeugt.

Die Synatx bei **PostgreSQL** sieht erwartungsgemäß etwas anders aus.

Syntax:

```
CREATE [ UNIQUE ] INDEX index_name ON table
     [ USING acc_name ] ( column [ ops_name ] [, ...] )
     | ( func_name( column [, ... ]) [ ops_name ] )
```

Beschreibung:

CREATE
Einleitendes Schlüsselwort für die CREATE Index-Anweisung.

UNIQUE
gibt an, dass ein eindeutiger Index verwendet wird, d.h. in der Schlüsselspalte dürfen keine Duplikate vorkommen. Beim Einfügen von Daten, die Duplikate liefern, wird eine Fehlermeldung ausgegeben.

index_name
Name des Indexes. Der Name muss eindeutig innerhalb der Tabelle, jedoch nicht innerhalb der DB sein.

table
Name der Tabelle, welche die Spalte enthält, die indiziert werden soll.

acc_name
Bezeichnung der Zugriffsmethode (BTREE, RTREE oder HASH) bei der Verwendung eines Index. Standardmäßig wird BTREE verwendet.

column
Spalte oder Spaltenliste, die indiziert wird.

ops_name
Eine zugehörige Operator-Klasse (z.B. box_ops oder bigbox_ops). Für jede zum Index gehörende Spalte kann eine Operator-Klasse definiert werden.

func_name
Eine Funktion, die einen indizierbaren Wert zurückgibt.

Es ist ratsam, mit weniger großen Indizes zu arbeiten, d.h. die Schlüsselwerte sollten knapp und eindeutig sein. Wird bei der Spaltenauswahl für die Indizierung Eindeutigkeit vorausgesetzt, so muss im Vergleich zu nicht eindeutigen Spaltenwerten, eine geringere Anzahl von Zeilen abgefragt werden. Es gilt sicherzustellen, dass für die Indexseiten, der benötigte Speicherplatz zur Verfügung steht. Im Allgemeinen basiert die Größe eines Index auf der Größe der Indexschlüssel, welche die Anzahl der Indexzeilen auf einer Indexseite und der Zeilen in der Tabelle bestimmt. Als angenäherten Richtwert kann man für einen gruppierten Index ca. 1 Prozent und bei nicht gruppierten Indizes ca. 30-40 Prozent der Tabellengröße nehmen. Nicht jeder Datenbankbenutzer hat das Recht, einen Index für die Tabellen zu erstellen. Je nach DBMS darf nur derjenige, der das Objekt erzeugt hat oder ein authorisierter Benutzer eine Indizierung vornehmen. Das Gleiche gilt für das Entfernen einer Indexstruktur. Mit der DROP-Anweisung können bestehende Indizes gelöscht werden.

Syntax:

```
DROP INDEX index_name [, ...]
```

Beschreibung:

DROP INDEX
Schlüsselwörter für die DROP INDEX-Anweisung.

index_name
Name des Indexes oder der Indexliste, die gelöscht werden soll.

Nach dem Löschen der Indexstruktur steht der zuvor belegte Festplattenspeicher wieder zur Verfügung. Wird eine Tabelle gelöscht, so werden auch die Indizes in dieser Tabelle gelöscht.

5.4.4 Vor- und Nachteile der Indexerstellung

Vielleicht ist man bei den funktionellen Möglichkeiten, die eine Indizierung bietet, geneigt, so viel Indizes wie möglich zu implementieren. Es sollten jedoch vor einer Indexerstellung zwei Faktoren berücksichtigt werden:

1. Welche Art von Daten sind in der Tabelle bzw. den zu indizierenden Spalten enthalten?
2. Welche Art von Abfrage wird auf dieser Tabelle ausgeführt?

Es sollten keine Spalten indiziert werden, die nur selten abgefragt werden oder die nur wenig eindeutige Werte haben. Einige *Datentypen* können von vornherein nicht indiziert werden (z.B. bit, image etc.). Primär- und Fremdschlüsselspalten sollten indiziert

werden, weil auf diese Spalten sehr häufig in Abfragen, gerade bei Verknüpfungen mit anderen Tabellen, zugegriffen wird. Generell sollte in Spalten mit Schlüsselwerten oder Daten, die häufig in lesender Weise verwendet werden, ein Index gesetzt werden.

Eine Erhöhung der Zugriffsgeschwindigkeit und die Formulierung einer speziellen Konsistenzbedingung kann durch Anlegen eines Index erzielt werden. Dabei gilt es zu beachten, dass *Geschwindigkeitsvorteile* beim Lesen durch erhöhten Platzaufwand und verringerte Geschwindigkeit bei manipulativen Zugriffen erkauft werden. Wie im Abschnitt »Wie werden Daten gespeichert?« gezeigt, werden beim Erstellen eines Index Indexseiten angelegt. Beim Zufügen neuer Datensätze in die Datenbank wird also auch neuer Platz auf den Indexseiten benötigt, der unter Umständen auch durch Neuerstellung von Indexseiten geschaffen werden muss. Selbst ein erneutes Sortieren der Zeiger auf den vorhandenen Indexseiten kann zu erheblichem Rechenaufwand führen. So stößt man also mit DELETE-, UPDATE- und INSERT-Anweisungen bei indizierten Spalten auf zusätzlichen Arbeitsaufwand. Allerdings kann eine DELETE- oder UPDATE-Anweisung mit zusätzlicher Suchabfrage durch Verwendung von Indizes einen Vorteil haben.

Der Nutzen eines Index hängt, mit dem Anteil des Abfrageergebnisses an den gesamten, von der Abfrage betroffenen Zeilen ab. Eine hohe *Selektivität* bedeutet eine größere Effizienz.

Beispiel 5.8:

Die Abbildung 5.9 zeigt die Vergabe von Indizes in der Tabelle *auftrag* aus der Beispieldatenbank. Schlüsselspalten, wie *kundennr*, *auftragnr* und *rechnnr*, sollten auf jeden Fall indiziert werden, weil über diese Verknüpfung mit anderen Relationen aufgebaut werden. Die Spalte *bezeichnung* eignet sich ebenfalls für eine Indizierung, da sie Informationen liefert die häufiger abgefragt werden. Die Spalte *auftragnr* ist mit dem primären Schlüssel belegt und steht daher sowieso als indizierte Spalte fest. Im Normalfall sollten Sie mit fünf bis acht Indizes pro Tabelle auskommen (siehe Abbildung 5.9).

Wenn in der Datenbank ein *Massenimport* vorgesehen ist und Tabellen daran beteiligt sind die mehrere Indizes beinhalten, so ist es sinnvoll, diese Indizes zunächst zu löschen und den Import durchzuführen. Später können die Indizes dann mit verhältnismäßig wenig Aufwand wieder erstellt werden. Das Gleiche gilt für Update-Vorgänge, bei denen ein großer Teil der Tabelle verändert wird.

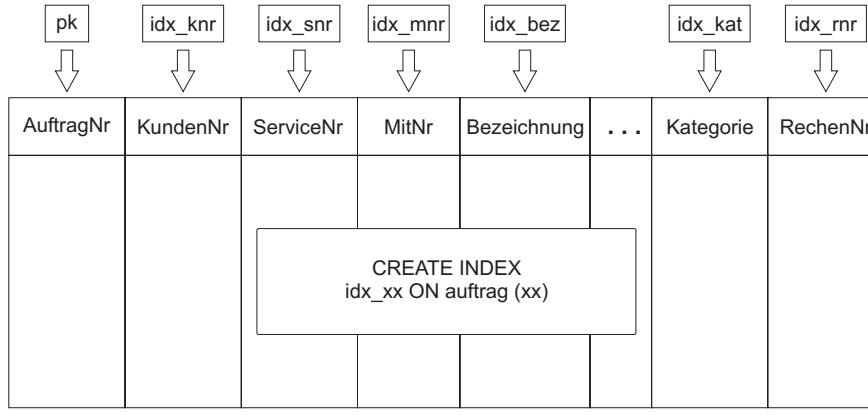


Abbildung 5.9: Indizes in der Beispieltabelle Auftrag

5.5 Sichten – die virtuellen Tabellen

Die Deutschen sind ein reiselustiges Volk. Jahr für Jahr pilgern Tausende in Urlaubsgebiete, die für viele gar nicht exotisch genug sein können. Trotzdem bleibt für viele ein Besuch am weißen Strand von Malibu oder bei den Cherubs Pyramiden in Ägypten ein unerfüllter Reisewunsch. Durch Reiseberichte in Büchern oder Videos haben sie dennoch die Möglichkeit einen kleinen vorgegebenen Ausschnitt dieser Schönheiten anzusehen.

Dem Datenbankbenutzer kann es durch Zugriffseinschränkungen verboten sein, auf sämtliche Daten einer Tabelle zuzugreifen. Über Sichten (Views) hat er jedoch die Möglichkeit, einen Teil der Tabelle einzusehen. So können nicht relevante oder geheime Daten für ihn ausgeblendet werden, ohne dass er überhaupt von ihrer Existenz weiß. Für einen einfachen Angestellten bleibt vielleicht die Gehaltshöhe seines Vorgesetzten eine stets unzugängliche Information, aber er kann seinem Chef eine nette Glückwunschkarte zum Geburtstag schicken, weil der Datenbankadministrator ihm die Sicht auf die Geburtstagsdaten des gesamten Personals erlaubt.

Eine Sicht kann demnach als eine »virtuelle Tabelle« verstanden werden, die ihre Daten aus einer oder mehreren so genannten Basistabellen bekommt. Physisch ist diese Tabelle gar nicht vorhanden, sondern wird bei jedem Aufruf neu generiert. Der Sinn sollte beim genannten Beispiel bereits angedeutet werden. Es gibt Daten, die je nach Benutzer und Anwendungsfall einfach nicht relevant sind und deshalb auch nicht »mitgeschleppt« werden sollten. Außerdem können auf diese Art und Weise Teilbereiche einer Tabelle vor unbefugtem Zugriff geschützt werden.

Mit der hinreichenden Sachlichkeit ausgedrückt, ermöglicht die Sicht eine vordefinierte Abfrage als Objekt in der Datenbank zu speichern.

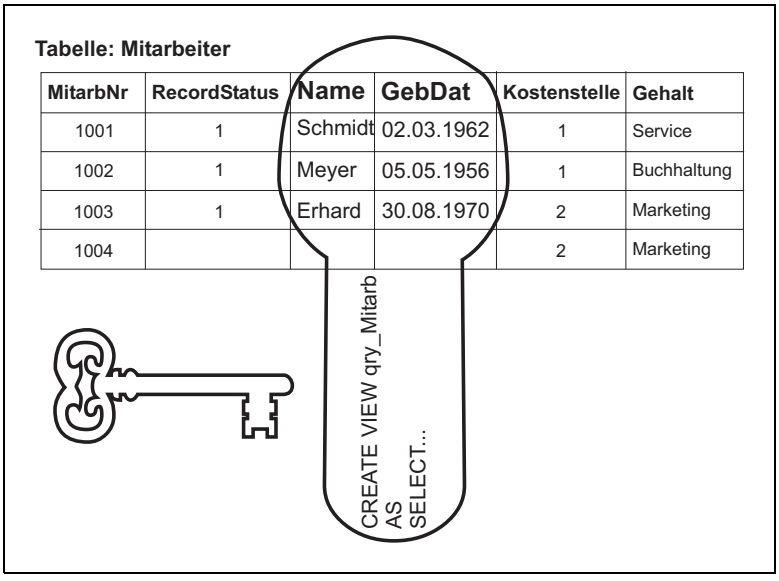


Abbildung 5.10: Beschreibung einer Sicht

Sichten werden oftmals verwendet für:

- eine Teilmenge von Zeilen oder Spalten einer Basistabelle,
- eine Union von zwei oder mehr Basistabellen,
- eine Verknüpfung von zwei oder mehr Basistabellen,
- eine statistische Zusammenfassung einer Basistabelle,
- eine Teilmenge einer anderen Sicht oder eine Kombination von Sichten und Basistabellen.

5.5.1 Erstellung einer Sicht

Eine Sicht kann entweder über Assistenten (MS Access oder MS SQL Server) oder aber über SQL-Anweisungen (PostgreSQL) erzeugt werden.

Hier soll nur der Weg über die entsprechende SQL-Anweisung demonstriert werden, weil dies die Vorgehensweise ist, die für alle RDBMS gültig ist. Sie können auch beim SQL Server oder bei Access die Sicht mit SQL erstellen. Der Name der Sicht muss dem Bezeichnungsreglement des DBMS entsprechen und darf nicht bereits verwendet sein. Für die Erstellung der Sicht benutzen sie also die angegebene Syntax.

Syntax:

```
CREATE VIEW view [ column [, ...] ]
   AS SELECT expression [ AS colname ] [, ...]
   FROM table [ WHERE condition ]
   [ WITH [ CASCADE | LOCAL ] CHECK OPTION ]
```

Beschreibung:

view
: Name der Sicht. Die Sichtnamen müssen den Regeln für Bezeichnung der DB-Objekte entsprechen.

column
: Durch Komma getrennte Liste der Namen für die Spalten in der Sicht.

SELECT-Anweisung
: Eine SELECT-Anweisung, die Spalten aus mehreren Tabellen und anderen Sichten enthalten kann.

WITH CHECK OPTION
: Bei dieser Option müssen alle Datenänderungen in der Sicht den in der SELECT-Anweisung festgelegten Kriterien entsprechen. Wenn dies nicht der Fall ist, wird die Änderung rückgängig gemacht.

CASCADE
: Überprüfung der Integrität in dieser Sicht und in den von ihr abhängigen Sichten.

LOCAL
: Überprüfung der Integrität in dieser Sicht.

Ein typisches Beispiel für den Einsatz einer Sicht könnte das »Sammeln« von Daten für einen Bericht sein. Dies soll anhand des folgenden Beispiels illustriert werden:

Beispiel 5.9:

In MS Access soll der Beispieldatenbank ein Bericht mit Kundendaten zugefügt werden. Der Bericht soll die Informationen *KundenNr*, *Kundenname*, *Strasse*, *PLZ*, *Ort* und *Telefonnummer* enthalten. Wenn wir uns das Datenbankdiagramm aus Kapitel 3 anschauen, sehen wir, dass die Informationen in den Tabellen *mitarbeiter*, *adresse* und *telekom* stecken. Der Sicht muss also eine Abfrage zugrunde liegen, welche die Tabellen miteinander verknüpft.

```
CREATE VIEW qryKundenbericht
  AS
  SELECT kunde.kundenNr, adresse.Name, adresse.plz,
     adresse.strassepostfach, adresse.ort, telekom.nummer
  FROM
```

```
    (adresse INNER JOIN telekom
     ON adresse.adressnr =  telekom.adressnr)
     INNER JOIN kunde
     ON adresse.adressnr = kunde.adressnr
WHERE adresse.typ = 'Kunde'
AND telekom.kategorie = 'Telefon';
```

5.5.2 Ändern und Löschen von Sichten

Verändert sich die Struktur der Basistabellen, so muss auch meistens die Definition der Sicht verändert werden. Die ALTER VIEW-Anweisung ändert die Definition einer Sicht und ermöglicht es, die Berechtigung für die Sicht beizubehalten.

Syntax für **MS SQL Server**:

```
ALTER VIEW view [ ( column [ ,...n ] ) ]
  AS
  SELECT expression [ AS colname ] [, ...]
  FROM table
  [ WHERE condition ]
  [ WITH CHECK OPTION ]
```

Die Bedeutung der Angaben entspricht im Wesentlichen denen der CREATE VIEW-Anweisung. Die Sicht vom Beispiel 5.9 wird in der WHERE-Klausel geringfügig verändert.

Beispiel 5.10:

```
ALTER VIEW qryKundenbericht
  AS
  SELECT kunde.kundenNr, adresse.Name, adresse.plz,
    adresse.strassepostfach, adresse.ort, telekom.nummer
    FROM (adresse INNER JOIN telekom
      ON adresse.adressnr = telekom.adressnr)
      INNER JOIN kunde
      ON adresse.adressnr = kunde.adressnr
    WHERE adresse.typ = 'Kunde'
      AND telekom.kategorie = 'Mobil';
```

Das Kriterium in der Bedingung wurde bei der Kategorie verändert. Statt der Telefonnummern sind nun die Mobil-Nummern aufgelistet.

Wird eine Sicht nicht mehr benötigt, so kann sie aus der Datenbank mit der DROP VIEW-Anweisung entfernt werden. Beim Löschen einer Sicht werden Definition und alle ihr zugewiesenen Berechtigungen gelöscht. Wird eine Basistabelle gelöscht, wird die darauf verweisende Sicht nicht automatisch gelöscht, sondern muss explizit gelöscht werden.

Syntax für **PostgreSQL**:

```
DROP VIEW view { RESTRICT | CASCADE }
```

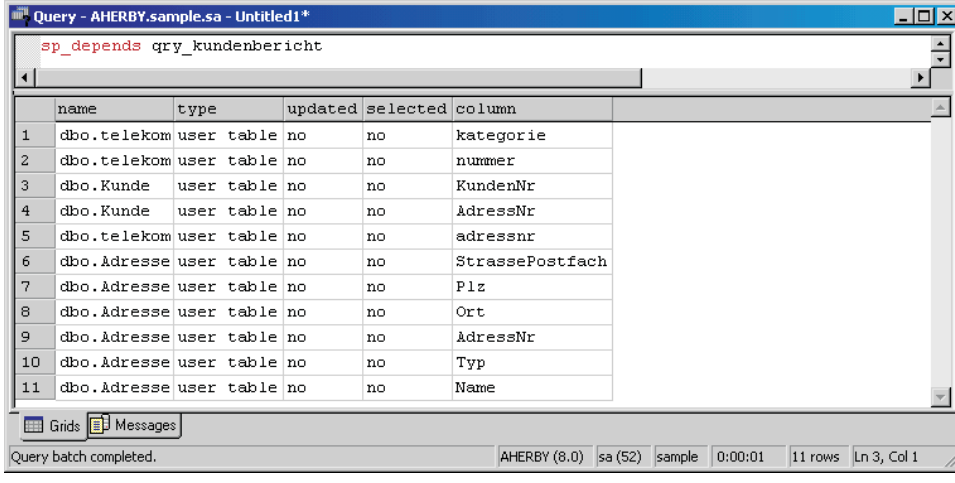
Beschreibung:

view
Name der zu löschenden Sicht.

RESTRICT
Es werden nur Sichten ohne Abhängigkeiten bzgl. anderer Sichten und Einschränkungen gelöscht.

CASCADE
Alle von der Sicht abhängigen Sichten und Einschränkungen werden ebenfalls gelöscht.

Die Optionen CASCADE und RESTRICT sind nicht in jedem DBMS Bestandteil des DROP VIEW-Konstruktes. Vor dem Löschen sollte man sich aber in jedem Fall über referenzierte Datenbankobjekte informieren.



	name	type	updated	selected	column
1	dbo.telekom	user table	no	no	kategorie
2	dbo.telekom	user table	no	no	nummer
3	dbo.Kunde	user table	no	no	KundenNr
4	dbo.Kunde	user table	no	no	AdressNr
5	dbo.telekom	user table	no	no	adressnr
6	dbo.Adresse	user table	no	no	StrassePostfach
7	dbo.Adresse	user table	no	no	Plz
8	dbo.Adresse	user table	no	no	Ort
9	dbo.Adresse	user table	no	no	AdressNr
10	dbo.Adresse	user table	no	no	Typ
11	dbo.Adresse	user table	no	no	Name

Abbildung 5.11: Anzeigen der Abhängigkeiten mit sp_depends

Beim MS SQL Server können mit der Systemprozedur *sp_depends* die Abhängigkeiten der Sicht *qryKundenbericht* analysiert werden. Die beteiligten Basistabellen mit den jeweiligen Spalten werden angezeigt.

5.5.3 Wann sollte mit Sichten gearbeitet werden?

Sichten bieten mehrere Vorteile und mit ihnen können einige wichtige Aufgaben ausgeführt werden. Sie verbergen die *Komplexität* von Daten in den einzelnen Tabellen, sie *konzentrieren Daten* für den Benutzer, es wird die Verwaltung von *Berechtigungen* vereinfacht, Daten werden für einen Bericht bereitgestellt und schließlich können Daten für den Export in andere Anwendungen *organisiert* werden.

Konzentration der Daten in einer Sicht

Das Beispiel 5.9 hat gezeigt, dass Daten, die für den Benutzer für die eine oder andere Aufgabe nicht relevant sind, in der Sicht einfach nicht aufgenommen werden. So wird nur der relevante Teil der Daten für den Benutzer sichtbar. Die zugrundeliegende Tabellenstruktur, die ihn zunächst nur verwirren würde, bleibt verborgen. Der Benutzer muss auch nicht mühsam die Abfragen erzeugen, die es ihm ermöglichen, Informationen zusammenzufassen oder Zusammenhänge zu erschließen. Die vordefinierten Abfragen in Form von Sichten bieten ihm die Funktionalität, auf die er nur zugreifen muss.

Verwaltung der Benutzerberechtigung wird durch Sichten vereinfacht

Sollen nur Teilbereiche von Tabellen bestimmten Benutzergruppen zugänglich gemacht werden, so können die entsprechenden Spalten in Sichten ausgewiesen werden. Der Administrator kann dann den Benutzern Berechtigungen für diese Sichten erteilen und muss nicht spaltenweise die Berechtigung in der Basistabelle vergeben. Auf diese Weise werden auch die Änderungen am Entwurf der zugrunde liegenden Basistabellen geschützt.

Daten für einen Bericht

In einem Bericht erscheinen nur selten sämtliche Informationen die in einer Basistabelle enthalten sind. Je nach Gebrauch, konzentriert man sich beim Bericht ganz gezielt auf einen Ausschnitt der Tabellen. Damit die Datenfülle nicht zu einer Papierflut führt, werden die notwendigen Daten durch die Sicht in einer Einheit zusammengefasst.

Export in andere Anwendungen

Daten können mittels Sichten auf Basis einer komplexen Abfrage zusammengefasst werden, um sie dann zur weiteren Analyse in eine andere Anwendung zu exportieren. Bestimmte Informationen der Tabelle *auftrag* aus der Beispieldatenbank werden für den Export in einem Statistikprogramm benötigt. Mit einer Sicht, die zusätzlich die Kundendaten bereitstellt, werden die Daten für den Export organisiert.

5.6 Gespeicherte Prozeduren – Umsetzen der Firmenlogik

Bisweilen reicht es nicht aus, die Integrität der Daten mit Einschränkungen zu erzwingen. Fehlerhafte Dateneingaben sollen nicht nur als Fehler angezeigt werden, sondern es soll direkt auf der Seite des Datenbankservers eine Korrektur vorgenommen werden. So sollen beispielsweise die Nebenkosten aus einer Fremdwährung in Euro-Beträge umgerechnet werden. Mit Hilfe von Einschränkungen kann das Format überprüft werden, aber eine entsprechende Umrechnung ist mit Einschränkungen nicht realisierbar. Immer wenn etwas mehr Logik benötigt wird, müssen kleine Programme diese Logik ausführen. Eine *gespeicherte Prozedur* (Stored Procedure) könnte ein solches Programm sein. Es handelt sich dabei um eine Auflistung von SQL-Anweisungen, die auf dem Server gespeichert sind. Sie unterstützen von Benutzern deklarierte Variablen, bedingte Ausführungen sowie weitere leistungsfähige Programmiermerkmale.

5.6.1 Funktionsweise

Gespeicherte Prozeduren entsprechen in mehrfacher Hinsicht den Prozeduren anderer Programmiersprachen. Es können Anweisungen enthalten sein, die Operationen in der Datenbank ausführen, d.h. z. B. auch andere Prozeduren aufrufen. Eingabeparameter können innerhalb der gespeicherten Prozedur bearbeitet werden und in Form von Ausgabeparametern an die aufrufende Stelle zurückgegeben werden. Außerdem kann an die aufrufende Stelle ein Statuswert zurückgegeben werden. So weiß die aufrufende Stelle, ob es sich um eine erfolgreiche oder fehlerhafte Ausführung gehandelt hat.

Geschäftsfunktionen können mit gespeicherten Prozeduren zusammengefasst werden. Geschäftsregeln oder Richtlinien können durch SQL-Anweisungen definiert werden. Bei Änderungen müssen diese nur einmal auf dem Datenbankserver geändert werden und nicht bei jedem Client. Die Logik ist somit auch unabhängig von der jeweiligen Anwendung, welche die Datenbank gerade ansteuert. Der Benutzereinblick in die Details der Datenbanktabellen wird verhindert, da durchzuführende Geschäftsfunktionen durch eine Reihe von gespeicherten Funktionen unterstützt werden. Der Datenverkehr im Netzwerk verringert sich auch durch Anwendung von gespeicherten Prozeduren, denn eine einzige SQL-Anweisung kann die Ausführung von einem Stapel an SQL-Anweisungen, welche auf dem Datenbankserver implementiert werden, auslösen.

5.6.2 Erstellen und Löschen gespeicherter Prozeduren

Die Vorgehensweise bei der Erstellung einer gespeicherten Prozedur ähnelt der bei den Sichten. Zunächst sollten die SQL-Anweisungen im einzelnen getestet werden. Dies kann z.B. beim MS SQL Server im Query Analyzer geschehen. Bei korrekter Syntax und Funktionalität kann die gespeicherte Prozedur erstellt werden.

CREATE PROCEDURE

Gespeicherte Prozeduren können mit der CREATE PROCEDURE-Anweisung erstellt werden. Die CREATE PROCEDURE-Anweisung kann nicht mit anderen SQL-Anweisungen im selben Stapel kombiniert werden. Innerhalb der gespeicherten Prozedur kann auf andere Tabellen, Sichten oder andere gespeicherte Prozeduren verwiesen werden. Nicht alle SQL-Anweisungen können innerhalb einer gespeicherten Prozedur ausgeführt werden, insbesondere solche, die neue Datenbankobjekte erzeugen.

Eine Schachtelung der gespeicherten Prozeduren ist auf mehreren Ebenen möglich. So kann eine Prozedur eine andere aufrufen, die wiederum eine andere aufruft.

Syntax für den **SQL Server**:

```
CREATE PROC [ EDURE ] procedure_name [ ; number ]
  [ { @parameter data_type }
   [ VARYING ] [ = default ] [ OUTPUT ]
  ] [ ,...n ]
  [ WITH
   { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION }
  ]
  [ FOR REPLICATION ]
  AS sql_statement [ ...n ]
```

Beschreibung:

CREATE PROCEDURE
Schlüsselwörter für diese Bezeichnung.

procedure_name
Name der neuen Prozedur. Der Name muss den Regeln für die Bezeichnung von DB-Objekten im DBMS entsprechen.

number
Optionale Angabe zur Erstellung einer Gruppe von Prozeduren mit dem gleichen Namen.

@parameter
Ein oder mehrere Übergabeparameter können innerhalb der Prozedur deklariert werden. Es handelt sich um lokale Parameter.

data_type
Datentyp des Übergabeparameters.

VARYING
Spezifikation für die als Ausgabeparameter unterstützte Ergebnismenge. Nur für Cursorparameter gültig.

default
Standardwert, der beim Prozeduraufruf für den Parameter verwendet wird.

OUTPUT
kennzeichnet die Parameter als Ausgabeparameter. Wert wird an die aufrufende Stelle zurückgegeben.

[,...n]
Platzhalter für weitere Parameter.

RECOMPILE
Die Prozedur wird bei jedem Aufruf neu kompiliert.

ENCRYPTION
Es wird verhindert, dass Unbefugte den Text der Prozedur einsehen können.

FOR REPLICATION
Die Prozedur wird ausschließlich für den Replikationsvorgang verwendet.

sql_statement
Eine oder mehrere SQL-Anweisungen, die beim Aufruf der Prozedur ausgeführt werden.

Bei der Erstellung einer gespeicherten Prozedur werden die enthaltenen Anweisungen auf ihre syntaktische Korrektheit überprüft. Der Name der Prozedur wird in einer Systemtabelle vom DBMS gespeichert. Es ist darauf zu achten, dass der Benutzer auch die notwendigen Rechte für die angesprochenen Datenbankobjekte hat. Der Besitzer der gespeicherten Prozedur und der Besitzer der zugrundeliegenden Tabelle sollten identisch sein. Das Limit für Anzahl der SQL-Anweisungen in der Prozedur und für die Schachtelung ist meist weit gespannt, trotzdem sollten die Grenzen hier nicht ausgetestet werden. Je komplexer die Anweisungen werden, desto höher ist auch die Fehleranfälligkeit. Für die Ausführung einer einzelnen Aufgabe sollte jeweils eine gespeicherte Prozedur verwendet werden. Die gespeicherten Prozeduren sollten zunächst auf dem Server erstellt und getestet, anschließend auf dem Client getestet werden.

Beispiel 5.11:

```
CREATE PROCEDURE euro_umrechnug @fremd_waehrung varchar(3) = 'DM', @betrag money,
@euro_wert money OUTPUT
AS
SET @euro_wert =
CASE @fremd_waehrung
     WHEN 'DM' THEN (@betrag/2)
     WHEN '$' THEN (@betrag/0.8765)
     WHEN 'YEN' THEN (@betrag/114.65)
END
```

In dem Beispiel 5.11 findet eine Währungsumrechnung statt. Der gespeicherten Prozedur kann beim Aufruf ein Betrag und das Kurzzeichen einer Fremdwährung übergeben werden. Wird keine Fremdwährung angegeben, so wird ein DM-Betrag angenommen (Standardwert). Innerhalb der Prozedur finden wir zunächst die Auswertung des Währungskennzeichens, realisiert mit einem CASE-Konstrukt. Ausgehend von der gewählten Währung findet dann die Umrechnung nach dem Schlüsselwort THEN statt. Der Rückgabewert *@euro_wert* ist mit dem Schlüsselwort OUTPUT versehen. Die END-Anweisung beendet die Bearbeitung im CASE-Konstrukt.

Wird die erstellte Prozedur nicht mehr benötigt oder soll sie neu erstellt werden, so kann sie mit der DROP PROCEDURE -Anweisung aus der Datenbank entfernt werden.

Syntax:

DROP PROCEDURE {procedure} [,...n]

Beschreibung:

DROP PROCEDURE
: Schlüsselwörter für diese SQL-Anweisung.

procedure
: Prozedur, die gelöscht werden soll.

[,...n]
: Platzhalter für weitere gespeicherte Prozeduren, die mit der selben Anweisung gelöscht werden sollen.

Die Prozedur aus dem Beispiel 5.11 könnte mit der folgenden Anweisung aus der Datenbank gelöscht werden:

Beispiel 5.12:

```
DROP PROC euro_umrechnug
```

Informationen über gespeicherte Prozeduren findet man auch in *Systemtabellen*. Es ist jedoch davon abzuraten, Einträge dort zu löschen. Bevor gespeicherte Prozeduren gelöscht werden, sollten vorhandene Abhängigkeiten überprüft werden. Dies kann z.B. mittels einer Systemprozedur geschehen, wie dies im nächsten Beispiel beim MS SQL Server getan wurde.

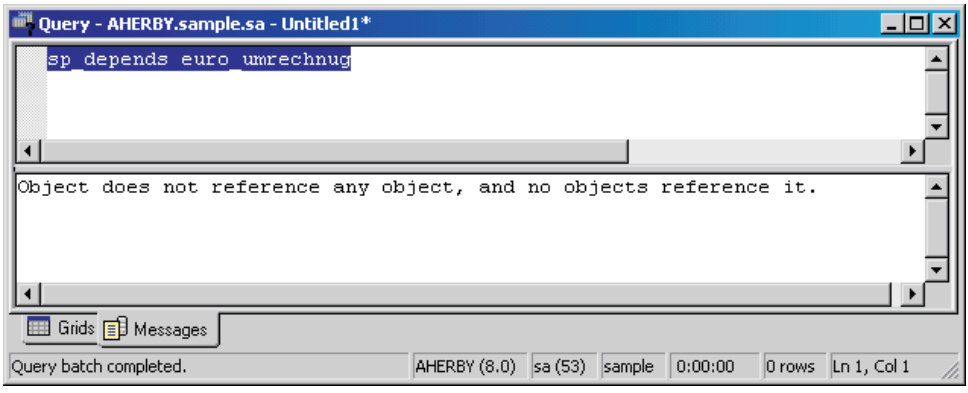


Abbildung 5.12: Funktion der Systemprozedur sp_depends

Die Systemprozedur *sp_depends* zeigt die vorhandenen Abhängigkeiten eines Datenbankobjektes an. Im Falle der gespeicherten Systemprozedur *euro_umrechnung* gibt es keine Abhängigkeiten, weil kein anderes Datenbankobjekt von der gespeicherten Prozedur aufgerufen wird.

5.6.3 Ausführen der gespeicherten Prozedur

Eine gespeicherte Prozedur kann allein oder als Teil einer INSERT-Anweisung ausgeführt werden. Mit der EXECUTE-Anweisung ist ein »selbstständiges« Ausführen möglich, wenn die Berechtigung dafür gegeben ist. Bei der erstmaligen Verarbeitung einer gespeicherten Prozedur wird diese zunächst erstellt und dann zum ersten Mal ausgeführt. Dadurch wird ihr Ausführungsplan in den *Prozedurcache* übernommen. Der Prozedurcache ist ein Seitenpool, der die Ausführungspläne für alle aktuell ausgeführten SQL-Anweisungen enthält. Die Größe des Prozedurcache ändert sich dynamisch entsprechend der *Aktivitätsstufe*. Der Prozedurcache befindet sich im Speicherpool, der Hauptspeichereinheit. Dort ist der größte Teil der Datenstrukturen, die Speichereinheiten benötigen, enthalten. (siehe Abbildung 5.13)

Beim erstmaligen Ausführen oder beim erneuten *Kompilieren* verarbeitet der *Abfrageprozessor* die gespeicherte Prozedur in einem Verfahren, das als *Auflösung* bezeichnet wird.

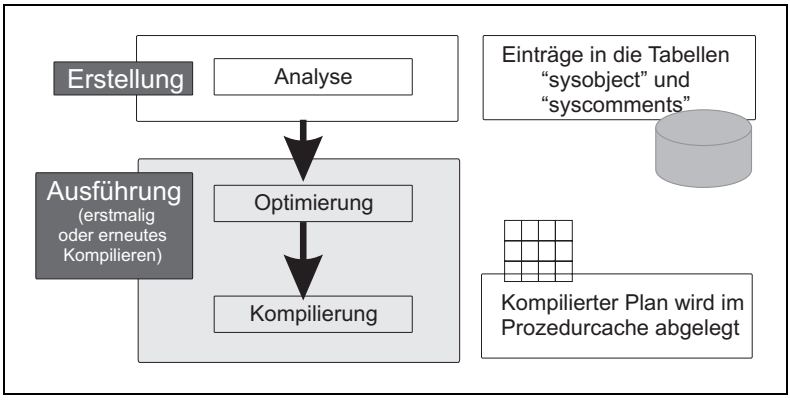


Abbildung 5.13: Ausführungsprozess einer gespeicherten Prozedur

Bestimmte Änderungen in einer Datenbank können dazu führen, dass ein Ausführungsplan nicht effizient eingesetzt wird oder nicht mehr gültig ist. Das DBMS erkennt diese Änderung und führt bei strukturellen Änderungen, Änderungen am Index oder Schlüssel automatisch eine erneute Kompilierung des Ausführungsplanes durch.

Die gespeichert Prozedur *euro_umrechnug* wird mit der EXECUTE-Anweisung ausgeführt.

Beispiel 5.13:

```
DECLARE @euro_wert money
EXECUTE euro_umrechnug 'DM', 34.50, @euro_wert OUTPUT
SELECT @euro_wert
```

Der Rückgabewert *@euro_wert* muss vor dem Aufruf der Prozedur deklariert werden. In der zweiten Zeile wird die Prozedur ausgeführt. Es wird ein Betrag von 34,50 DM zur Umrechnung übergeben. Der berechnete Wert, der zurückgegeben wird, steckt zunächst in der lokalen Variablen *@euro_wert* und wird mit der SELECT-Anweisung angezeigt.

Mit der INSERT-Anweisung kann eine Ergebnismenge, die von einer lokalen gespeicherten Prozedur zurückgegeben wird, in eine lokale Tabelle eingefügt werden. Die Tabelle muss bereits vorhanden sein und die Datentypen müssen übereinstimmen.

Beispiel 5.14:

```
CREATE PROCEDURE select_temp1
AS
SELECT 1, bezeichnung, kundennr, auftragnr, anzstunden
FROM temp1
```

Die gespeicherte Prozedur *select_temp1* beinhaltet in diesem Beispiel nichts anderes als eine SELECT-Anweisung mit der zugrundeliegenden Tabelle temp1. Es werden keine Übergabeparameter verwendet. Statt der gespeicherten Prozedur könnte hier auch eine Sicht verwendet werden. Der nächste Schritt soll nun das Aufrufen dieser Prozedur innerhalb einer INSERT-Anweisung sein.

Beipiel 5.15:

```
INSERT INTO service (recordstatus, bezeichnung, kundennr, auftragnr, anzstunden)
EXECUTE select_temp1
```

Die INSERT-Anweisung ist, wie bereits bekannt, mit der Angabe der Zieltabelle und der Zielspalten aufgebaut. Es wird jedoch keine SELECT-Anweisung und auch keine Werteliste zum Einfügen neuer Werte benutzt, sondern die Ergebnismenge, welche die gespeicherte Prozedur liefert.

5.6.4 Ändern von gespeicherten Prozeduren

Datenbankobjekte müssen aufgrund von Veränderungen an der Struktur der Objekte, die sie abbilden, ebenfalls geändert werden. Eine Veränderung, die an einer Tabelle vorgenommen werden muss, kann auch eine Modifizierung der gespeicherten Prozedur erfordern. Mit ALTER PROCEDURE können vorhandene gespeicherte Prozeduren verändert werden.

Syntax:

```
ALTER PROC [ EDURE ] procedure_name [ ; number ]
 [ { @parameter data_type }
  [ VARYING ] [ = default ] [ OUTPUT ] ] [ ,...n ]
 [ WITH
 { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
 [ FOR REPLICATION ]
AS
 sql_statement [ ...n ]
```

Die einzelnen Klauseln wurden bereits zur CREATE PROCEDURE-Anweisung beschrieben. Geschachtelte Prozeduren sind von der ALTER PROCEDURE-Anweisung nicht betroffen.

Beispiel 5.16:

```
ALTER PROCEDURE select_temp1
AS
SELECT 1, bezeichnung, kundennr, auftragnr, anzstunden, SUBSTRING(USER,1,10),
SUBSTRING(USER,1,10)
FROM temp1
```

Die gespeicherte Prozedur *select_temp1* wird um zwei Spalten in der Ergebnismenge erweitert. Die Funktion USER liefert den Namen des aktuellen Datenbankbenutzers. Mit der Funktion SUBSTRING wird ein Teilstring vom Namen extrahiert.

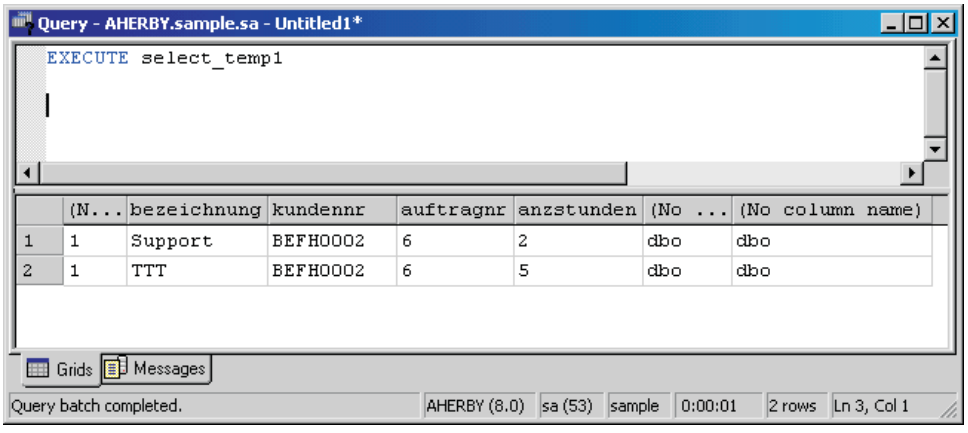


Abbildung 5.14: Ausführung einer gespeicherten Prozedur

Die Abbildung 5.14 zeigt das Ergebnis nach Ausführung der gespeicherten Prozedur *select_temp1*. Die Funktion USER hat, wie die Abbildung zeigt, als Ergebnis den Datenbankbesitzer *dbo* ausgewiesen. *DBO* ist ein Benutzer, der die Rechte eines Systemadministrators hat.

5.7 Zusätzliche Funktionalität durch Trigger

Trigger sind, ähnlich den Datenbankprozeduren, kleine SQL-Programmeinheiten. Man beschreibt Trigger auch gern als »Sonderfall der gespeicherten Prozeduren«. Im Gegensatz zu den gespeicherten Prozeduren, werden die Trigger nicht bewusst vom Anwender oder Programmierer aufgerufen, sondern durch ein Ereignis. Trigger werden für eine bestimmte Tabelle definiert, die als *Triggertabelle* bezeichnet wird. Bestimmte Aktionen in dieser Tabelle rufen Trigger auf, welche zuvor in Form von SQL-Anweisungen definierte Aufgaben durchführen.

5.7.1 Nutzbarkeit von Datenbanktriggern

Eine Verwendungsmöglichkeit von Triggern liegt in der Aufrechterhaltung der Datenintegrität. Dies wurde bereits im Abschnitt 5.2 erläutert. Der Vorteil gegenüber Einschränkungen, die ebenfalls zur Erhaltung der Datenintegrität verwendet werden, besteht in der *erweiterten Verarbeitungslogik*, die ein Trigger enthalten kann. Durch einen

Trigger, der bei jedem Löschvorgang einer Tabelle aktiviert wird, kann beispielsweise die referenzielle Integrität überprüft werden und durch eine DELETE-Anweisung in Referenztabellen aufrechterhalten werden. Man nennt diesen Vorgang auch »Kaskadieren von Änderungen« über verknüpfte Tabellen. Zusätzlich kann Dateneingabe nicht nur formal gecheckt werden, sondern auch logisch und sogar mit bereits vorhandenen Daten in der Datenbank verglichen werden.

Beispiel 5.17:

Ein Patient nimmt an einer klinischen Studie teil. In regelmäßigen Abständen bekommt er ein Medikament verabreicht. Vor und nach Einnahme des Medikamentes werden Blutdruck-, Blutwert- und EKG-Messungen durchgeführt. Bei gewissen Bereichsüberschreitungen darf dem Patienten das Medikament nicht gegeben werden. Gibt der Arzt nun die entsprechenden Werte in einer Studiendatenbank ein, so kann mittels eines Triggers überprüft werden, ob die verabreichte Dosis, für die zuvor gemessenen Werte ordnungsgemäß war und ob sie den Therapierichtlinien entspricht. Dem Arzt kann direkt eine Warnmeldung ausgegeben werden und eingesetzte Prüfstellen können online informiert werden. Die Ausgabe oder Weiterleitung der Warnung könnte Bestandteil der Logik innerhalb eines Triggers sein.

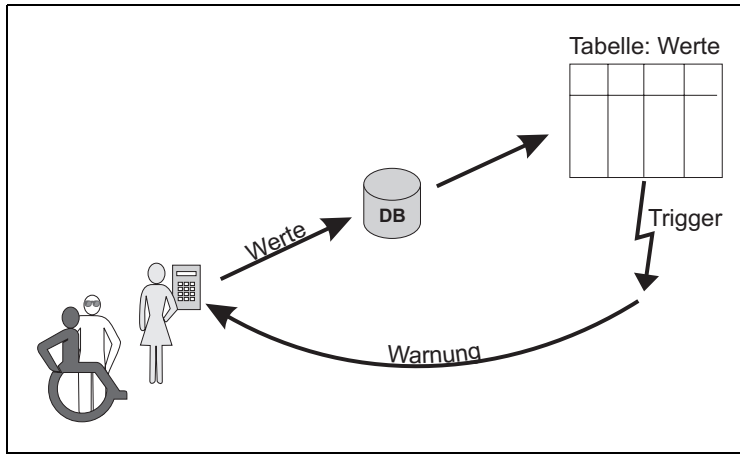


Abbildung 5.15: Anwendungsfall für einen Datenbanktrigger

Es können neben Eingabechecks auch benutzerdefinierte Fehlermeldungen ausgegeben werden. Dadurch kann der Status von Aktionen dynamisch überwacht werden.

Mit Hilfe von Triggern ist es möglich, einen *Statusvergleich* vor und nach Änderungen an den Tabellen vorzunehmen. So kann bei Veränderung von wichtigen Informationen der alte Zustand der Daten in einer Archivierungstabelle aufgezeichnet werden. Die erforderlichen Anweisungen wären auch hier wieder Bestandteil der Triggerlogik.

5.7.2 Funktionsweise von Triggern

Trigger sind *reaktiv*, d.h. sie werden ausgeführt wenn INSERT-, UPDATE- oder DELETE-Anweisungen für die Tabelle, für die der Trigger definiert ist, durchgeführt werden. Für eine einzelne Tabelle können normalerweise mehrere Trigger definiert werden. Für die Auslösung der Trigger existiert dann keine bestimmte Reihenfolge. Eine INSERT-, UPDATE- oder DELETE-Aktion, durch die ein Trigger ausgelöst wird, kann mehrere Zeilen betreffen.

INSERT-Trigger

Wird ein INSERT-Trigger ausgelöst, so wird sowohl in der Triggertabelle, als auch in der *Protokolltabelle inserted* eine neue Zeile hinzugefügt. Die Protokolltabelle *inserted* enthält eine Kopie des eingefügten Datensatzes. Werden durch den INSERT-Befehl mehrere Zeilen eingefügt, so sind diese alle in *inserted* aufgelistet. Die Tabelle *inserted* kann nur innerhalb des Triggers abgefragt werden und zwar ebenso, als wäre sie eine Kopie der Triggertabelle *telekom* mit gleicher Struktur. Bei einer weiteren INSERT-Anweisung kann nicht auf den Inhalt der Tabelle *inserted* zugegriffen werden, welcher durch die vorherigen Anweisungen zustande kam. Bei dieser Protokolltabelle handelt es sich um eine logische Tabelle.

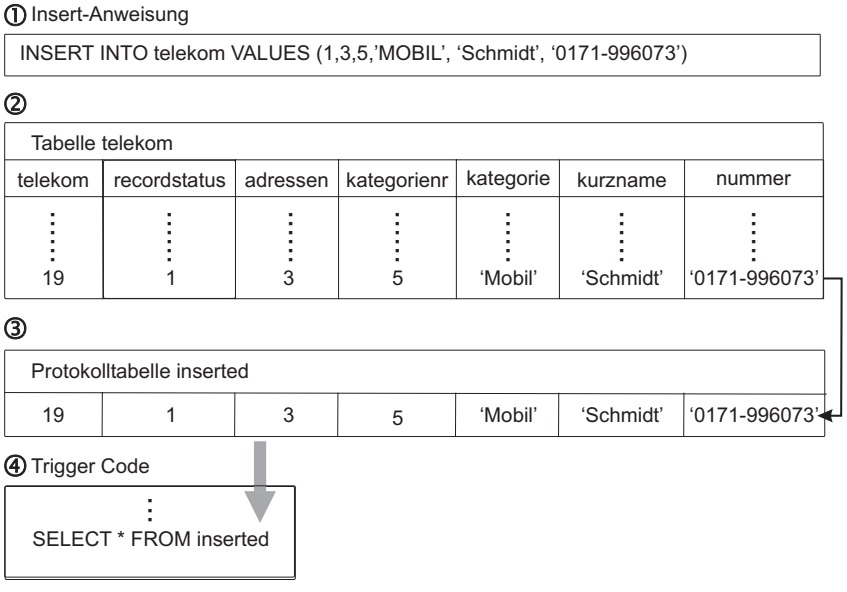


Abbildung 5.16: Beschreibung eines INSERT-Triggers

DELETE-Trigger

Wenn ein DELETE-Trigger ausgelöst wird, sind die gelöschten Zeilen in der Protokolltabelle *deleted* abrufbar. Diese Protokolltabelle ist wie die Tabelle *inserted* eine logische Tabelle mit Struktur der Triggertabelle *telekom* (siehe Abbildung 5.17). Die Datensätze sind in der Triggertabelle bereits gelöscht, wenn sie in die Protokolltabelle angehängt werden.

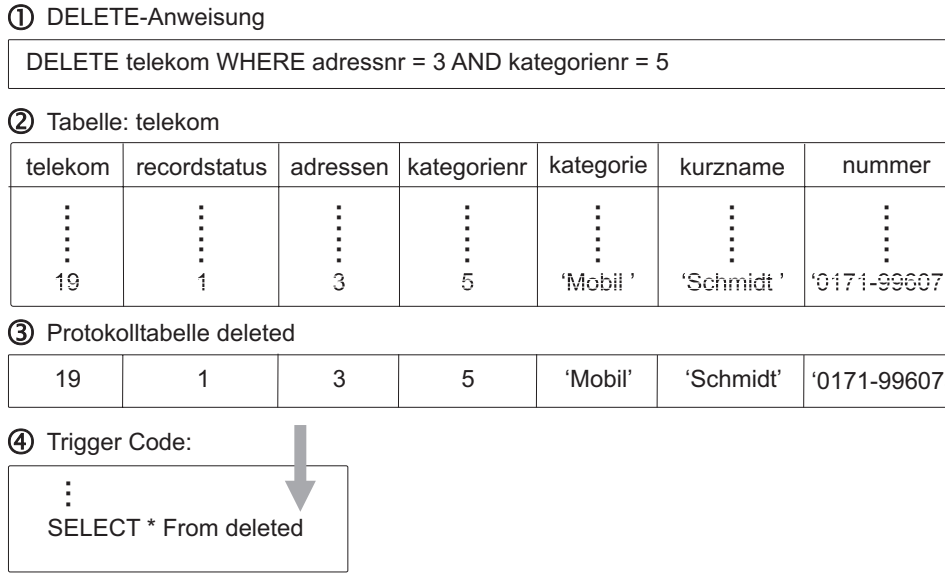


Abbildung 5.17: Beschreibung eines DELETE-Triggers

UPDATE-Trigger

Bei der UPDATE-Anweisung finden wir zwei Protokolltabellen, die vom UPDATE-Trigger ausgewertet werden können. In der Tabelle *inserted* ist der Zustand nach und in der Tabelle *deleted* der Zustand vor der Ausführung der UPDATE-Anweisung. Für *inserted* und *deleted* gelten die gleichen Bedingungen, wie sie beim DELETE- und beim INSERT-Trigger beschrieben wurden. Es kann ein UPDATE-Trigger für eine gesamte Tabelle oder auch für einzelne Spalten erstellt werden.

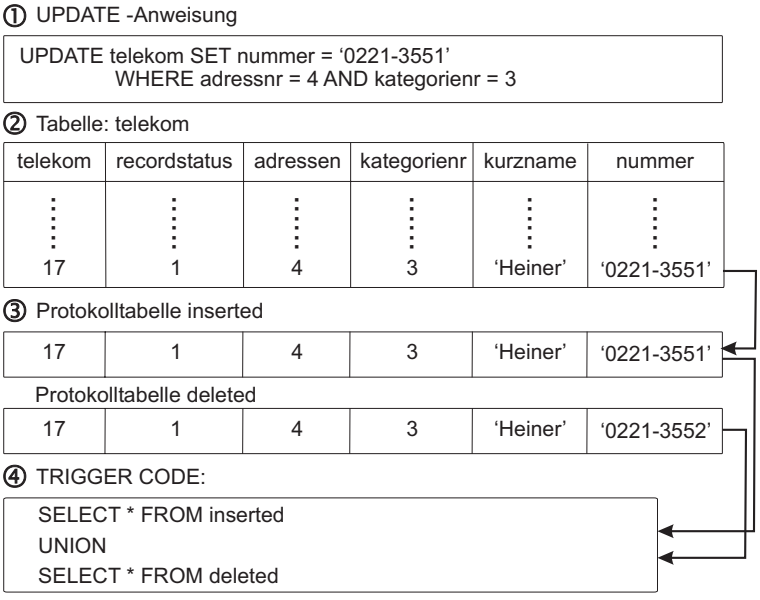


Abbildung 5.18: Beschreibung eines UPDATE-Triggers

5.7.3 Erstellen eines Triggers

Mit der Anweisung CREATE TRIGGER können Trigger erstellt werden. Trigger können nur durch den Tabellenbesitzer erstellt werden. Es ist ebenso möglich für Sichten einen Trigger zu erstellen. In der Triggerlogik kann ebenfalls auf Sichten und temporäre Tabellen verwiesen werden. Wie bei einer gespeicherten Prozedur können die SQL-Anweisungen eine Ergebnismenge zurückgeben, was jedoch im Normalfall nicht geschieht, da sich ansonsten die Arbeit an und mit der Datenbank verzögern würde. Man nehme nur an, dass bei jeder Änderung an einer Tabelle der alte und neue Zustand der Daten angezeigt wird. Das wäre für den Anwender höchst lästig. Die Triggerlogik läuft meist »still und heimlich« im Hintergrund ab, ohne dass der Datenbankbenutzer etwas davon merkt.

Syntax für **PostgreSQL**:

```
CREATE TRIGGER name { BEFORE | AFTER } { event [OR ...] }
ON table FOR EACH { ROW | STATEMENT }

EXECUTE PROCEDURE func ( arguments )
```

Beschreibung:

name
Name des Triggers.

table
Name der Triggertabelle.

event
Ereignis, bei dem der Trigger ausgelöst wird entweder INSERT, DELETE oder UPDATE.

func
benutzerdefinierte Funktion.

Syntax bei **MS SQL Server**:

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
    { { FOR | AFTER | INSTEAD OF }
        { [ DELETE ] [ , ] [ INSERT ] [ , ] [ UPDATE ] }
        [ WITH APPEND ]
        [ NOT FOR REPLICATION ]
        AS
        sql_statement [ ...n ]
    }
    |
    { ( FOR | AFTER | INSTEAD OF ) { [ INSERT ] [ , ] [ UPDATE ] }
        [ WITH APPEND ]
        [ NOT FOR REPLICATION ]
        AS
        { IF UPDATE ( column )
            [ { AND | OR } UPDATE ( column ) ]
                [ ...n ]
        | IF ( COLUMNS_UPDATED ( ) { bitwise_operator } updated_bitmask )
                { comparison_operator } column_bitmask [ ...n ]
        }
        sql_statement [ ...n ]
    }
}
```

Beschreibung:

trigger_name
Name des Triggers.

table | *view*
Name der Tabelle oder der Sicht, in welche der Trigger implementiert werden soll.

WITH ENCRYPTION
Es wird verhindert, dass Unbefugte den Text des Triggers einsehen können.

AFTER
Der Trigger wird nur ausgeführt, wenn die zum Aufruf führenden Aktionen erfolgreich waren.

INSTEAD OF
Die SQL-Anweisung im Trigger überschreibt die Änderungen der Anweisung, die ihn auslöste.

WITH APPEND
Ein bereits existierender Trigger wird zusätzlich ausgeführt. Diese Klausel ist aus Gründen der Abwärtskompatibilität vorhanden.

NOT FOR REPLICATION
Der Trigger wird nicht durch den Replikationsvorgang ausgelöst.

UPDATE (column)
Spezifizierung einer bestimmten Spalte. Nicht bei DELETE möglich.

bitwise_operator
Bit-Operator, der für den Vergleich benötigt wird.

updated_bitmask
Eine Bitmaske vom Typ *integer* der Spalten, die eingefügt oder verändert wurden.

comparison_operator
Vergleichsoperator.

column_bitmask
Bitmaske zur Überprüfung, ob Spalten geändert oder eingefügt wurden.

sql_statement
Eine oder mehrere SQL-Anweisungen mit der auszuführenden Logik.

Ein Trigger kann nur in der aktuellen Datenbank erstellt werden, er kann sich aber auf Objekte außerhalb der Datenbank beziehen. Es gibt einige SQL-Anweisungen (CREATE DATABASE, RESTORE DATABASE, RESTORE LOG etc.), insbesondere DCL-Anweisungen, die nicht in einem Trigger verwendet werden dürfen. In einigen DBMS sind multiple Trigger möglich. Eine Tabelle kann für die gleiche Aktion mehrere Trigger haben. So kann beispielsweise die Tabelle *service* der Beispieldatenbank mehrere DELETE-Trigger haben, die unterschiedliche Aufgaben ausführen.

Beispiel 5.18:

```
CREATE TRIGGER trg_delkategorie
ON kategorie
FOR DELETE
AS
PRINT 'Achtung Datensatz wurde gelöscht!'
```

Das Beispiel 5.18 zeigt zunächst einen einfachen Anwendungsfall. Die Tabelle *kategorie* enthält Definitionen unterschiedlicher Objekte. Üblicherweise werden Datensätze aus dieser Tabelle nur nach Absprache mit der Administration gelöscht. Man sollte die Berechtigung zum Löschen also nur einem bestimmten Personenkreis überlassen. Selbst für diesen eingeschränkten Personenkreis soll eine Warnmeldung ausgegeben werden. Der Trigger *trg_delkategorie* erfüllt diese Aufgabe. Immer wenn in der Tabelle *kategorie* ein Datensatz gelöscht wird, erscheint die Nachricht »Achtung Datensatz wurde gelöscht!«. Man kann sich nun fragen, wie sinnvoll diese Meldung ist, wenn der Datensatz sowieso schon gelöscht ist. Vielleicht wäre es hilfreich, den gelöschten Datensatz in einer Archivierungstabelle zu speichern. Dies passiert in dem nächsten Beispiel:

Beispiel 5.19:

```
CREATE TRIGGER trg_archkategorie
ON kategorie
FOR DELETE
AS
INSERT INTO archkategorie
SELECT * FROM deleted
```

Der Trigger *tgr_archkategorie* archiviert die gelöschten Datensätze aus der Tabelle *kategorie* in einer Archivierungstabelle *archkategorie*. Dabei wird einfach der oder die gelöschten Zeilen aus der Protokolltabelle *deleted* selektiert und über eine INSERT-Anweisung in die Tabelle *archkategorie* eingefügt.

Mit der folgenden Anweisung können beide Trigger getestet werden:

Beispiel 5.20:

```
DELETE FROM kategorie WHERE kategorie = 18;
```

Manche DBMS bieten die Möglichkeit, jedem Datensatz einer Tabelle automatisch eine *TIMESTAMP* (Zeitstempel) mitzugeben. Dabei wird aktuelles Datum und Zeit in eine Spalte der Tabelle geschrieben. Dort, wo diese Funktionalität nicht standardmäßig implementiert ist, kann sie sehr einfach mit Hilfe eines INSERT-Triggers realisiert werden.

Beispiel 5.21:

```
CREATE TRIGGER trg_instelekom
ON telekom
FOR INSERT
AS
UPDATE telekom SET createdate = GETDATE()
WHERE telekomnr =
(SELECT telekomnr FROM inserted)
```

In der Tabelle *telekom* wird beim Einfügen eines neuen Datensatzes die Protokolltabelle *inserted* ausgewertet, indem die *telekomnr* des neu eingefügten Satz abgefragt wird. In der UPDATE-Anweisung wird der Zeile mit der gefundenen *telekomnr* das aktuelle Systemdatum und die Uhrzeit mittels der Funktion GETDATE() geschrieben. Dieser Trigger kann mit einer INSERT-Anweisung für die Tabelle *telekom* überprüft werden.

5.7.4 Ändern und Löschen von Triggern

Dem aufmerksamen Leser ist bei dem letzten Beispiel vielleicht aufgefallen, dass der Trigger *trg_instelekom* einen Schwachpunkt hat. Was passiert, wenn mehrere Datensätze gleichzeitig eingefügt werden? Die Unterabfrage würde mehrere Werte liefern, was in der UPDATE-Anweisung zu einem Fehler führen würde. Dieser Trigger muss also optimiert werden. Mit der ALTER TRIGGER-Anweisung kann dies geschehen.

Syntax für den **SQL Server**:

```
ALTER TRIGGER trigger_name
ON ( table | view )
[ WITH ENCRYPTION ]
{
    { ( FOR | AFTER | INSTEAD OF )
     { [ DELETE ] [ , ] [ INSERT ] [ , ] [ UPDATE ] }
        [ NOT FOR REPLICATION ]
        AS
        sql_statement [ ...n ]
    }
    |
    { ( FOR | AFTER | INSTEAD OF )
     { [ INSERT ] [ , ] [ UPDATE ] }
        [ NOT FOR REPLICATION ]
        AS
        { IF UPDATE ( column )
        [ { AND | OR } UPDATE ( column ) ]
        [ ...n ]
        | IF ( COLUMNS_UPDATED ( )
             { bitwise_operator } updated_bitmask )
        { comparison_operator } column_bitmask [ ...n ]
        }
        sql_statement [ ...n ]
    }
}
```

Die einzelnen Angaben wurden bereits bei der Anweisung CREATE TRIGGER beschrieben. Durch die geänderte Definition wird die Definition des vorhandenen Triggers durch die neue Definition ersetzt.

Beispiel 5.22:

```
ALTER TRIGGER trg_instelekom
ON telekom
FOR INSERT
AS
UPDATE telekom SET createdate = GETDATE()
WHERE telekomnr IN
(SELECT telekomnr FROM inserted)
```

Statt des Gleichheitszeichens wurde das Schlüsselwort IN verwendet. Damit wird berücksichtigt, dass die Unterabfrage mehrere Werte liefern kann, die allesamt in der UPDATE-Anweisung bearbeitet werden.

Die ALTER TRIGGER-Anweisung steht bei PostgreSQL nicht zur Verfügung. Hier muss zunächst die DROP TRIGGER-Anweisung verwendet werden, um den Trigger zu löschen und schließlich mit CREATE TRIGGER wieder einzufügen.

Syntax für **PostgreSQL**:

```
DROP TRGGER trigger_name ON table
```

Syntax für **MS SQL Server**:

```
DROP TRIGGER trigger_name
```

Beispiel 5.23:

```
DROP TRIGGER trg_instelekom;
```

Manchmal ist es auch gewünscht, den Trigger nicht zu löschen, sondern ihn nur für einen gewissen Zeitraum zu *deaktivieren*. Dies kann z.B. beim Massenimport oder in der Testphase sehr hilfreich sein. Leider bieten nicht alle DBMS diese Option, deshalb müssen sie sich zunächst in der Dokumentation zum DBMS darüber informieren.

5.7.5 Rekursive Trigger

Jeder Trigger kann eine UPDATE-, INSERT- oder DELETE-Anweisung enthalten, die dieselbe Tabelle oder eine andere Tabelle betrifft. Ein Trigger kann sich durch Änderungen, die er selbst verursacht hat, nochmals aktivieren. Diese *Selbstaktivierung* nennt man auch *rekursiven Vorgang*. Rekursive Trigger stellen eine komplexe Funktion dar, mit deren Hilfe Sie komplexere Beziehungen, wie z.B. auf sich selbst verweisende Beziehungen auflösen können.

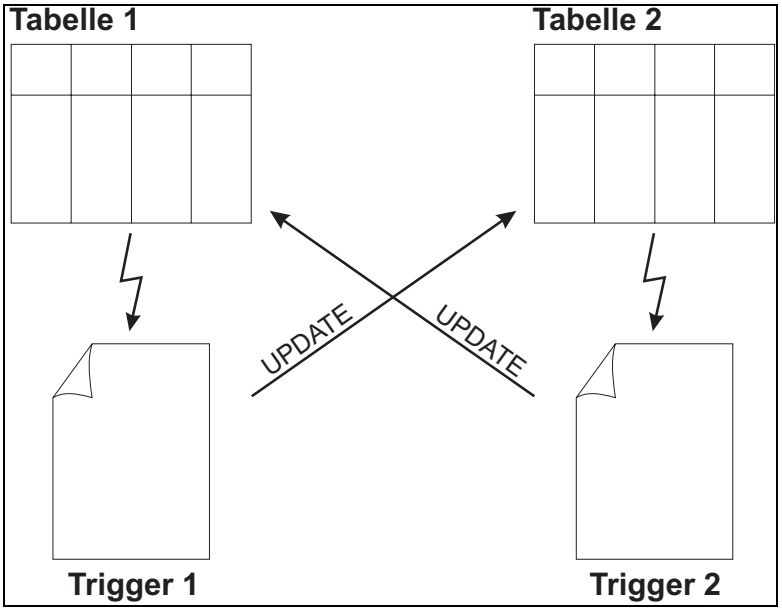


Abbildung 5.19: Darstellung rekursiver Trigger

Auch eine Konstellation, wie sie in Abbildung 5.19 gezeigt ist, wäre möglich. Der Trigger *Trig1* wird durch eine Änderung in der Tabelle *Tab1* aktiviert und modifiziert die Tabelle *Tab2*. Die Tabelle *Tab2* löst bei der Veränderung den Trigger *Trig2* aus, der wiederum die Tabelle *Tab1* verändert. Wenn durch einen Trigger in einer rekursiven Schleife eine Endlosschleife ausgelöst wird, so wird dies bis zu einer gewissen Schachtelungsebene ausgeführt und schließlich vom DBMS mit einem Rollback abgebrochen.

5.8 Skripterstellung

Ein Datenbankskript ist eine Anzahl von einer oder mehreren SQL-Anweisungen, bei deren Ausführung bestimmte Aufgaben erledigt werden. Für die Implementierung einer Datenbank mit all ihren Objekten, ja sogar mit dem Import von Daten, wird typischerweise ein Datenbankskript verwendet. Dieses Skript kann in einem simplen Texteditor erstellt werden und später im DBMS geladen und ausgeführt werden. Einige DBMS haben jedoch auch für die Skripterstellung Hilfswerkzeuge, die durchaus genutzt werden sollten, um sich die Arbeit zu erleichtern. Einmal erstellte und getestete Datenbankskripte können immer wieder eingesetzt werden, um eine Datenbank neu zu erstellen. Dadurch können aufwändige Entwicklungs- und Testphasen zeitlich verringert werden. Das SQL-Skript kann zunächst auf einem Testsystem entwickelt und verändert werden und erst nach erfolgreicher Testreihe wird das Skript im Pro-

duktivsystem ausgeführt. Die Vorteile, die sich hieraus ergeben, sind minimale Ausfallzeiten und die Tatsache, dass Änderungen auch von weniger geschultem Personal ausgeführt werden können.

5.8.1 Vom SQL-Skript zur Datenbank

Ein Skript kann verwendet werden, um sämtliche Aufgaben auszuführen, die notwendig sind, um eine Datenbank aufzusetzen. Skripte können jedoch auch in verschiedene logische Einheiten aufgeteilt werden, um einzelne Objekte zu implementieren. So kann ein Skript für die Erstellung der Datenbank, das nächste für Tabellen und wieder ein anderes für Triggerimplementierung verwendet werden. Welches Vorgehen sinnvoller ist, hängt von der Komplexität der Datenbank ab. Für eine »kleine« Datenbank mit vielleicht nur fünf Tabellen und wenig zusätzlichen Elementen reicht ein Skript sicherlich aus. Notwendigerweise muss hier eine Reihenfolge bei der Objekterstellung eingehalten werden:

1. Datenbank erstellen,
2. Benutzerdefinierte Datentypen erzeugen,
3. Definition von Konstanten,
4. Tabellen anlegen,
5. Import von Daten,
6. Erstellen von Indizes,
7. Einschränkungen erzeugen,
8. Sichten erzeugen,
9. Schlüssel implementieren,
10. Gespeicherte Prozeduren,
11. Trigger erstellen.

Skripte sollten, auch wenn sie sehr klein sind, zumindest in den ersten Zeilen ein wenig Dokumentation beinhalten. Es reicht, stichwortartig die Funktion der Anweisungen zu beschreiben. Ist die Skriptdatei erstellt und ihre korrekte Funktion getestet, so kann die Datei mit der Dateierweiterung .SQL gesichert werden (siehe Abbildung 5.20).

Das Skript für die Beispieldatenbank finden Sie auf der Begleit-CD im Verzeichnis »Skripte«.

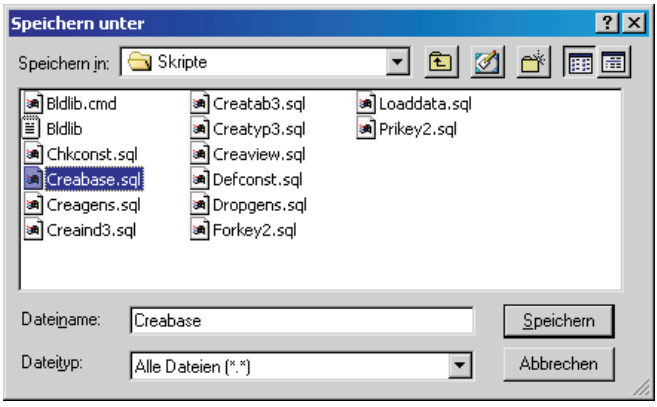


Abbildung 5.20: Speichern einer Skriptdatei

5.8.2 Skript ausführen

Ist bereits ein Skript vorhanden, kann es in der Ausführungseinheit, beim SQL Server ist es der Query Analyzer, geladen werden. Dort kann es ausgeführt oder wenn nötig geändert werden.

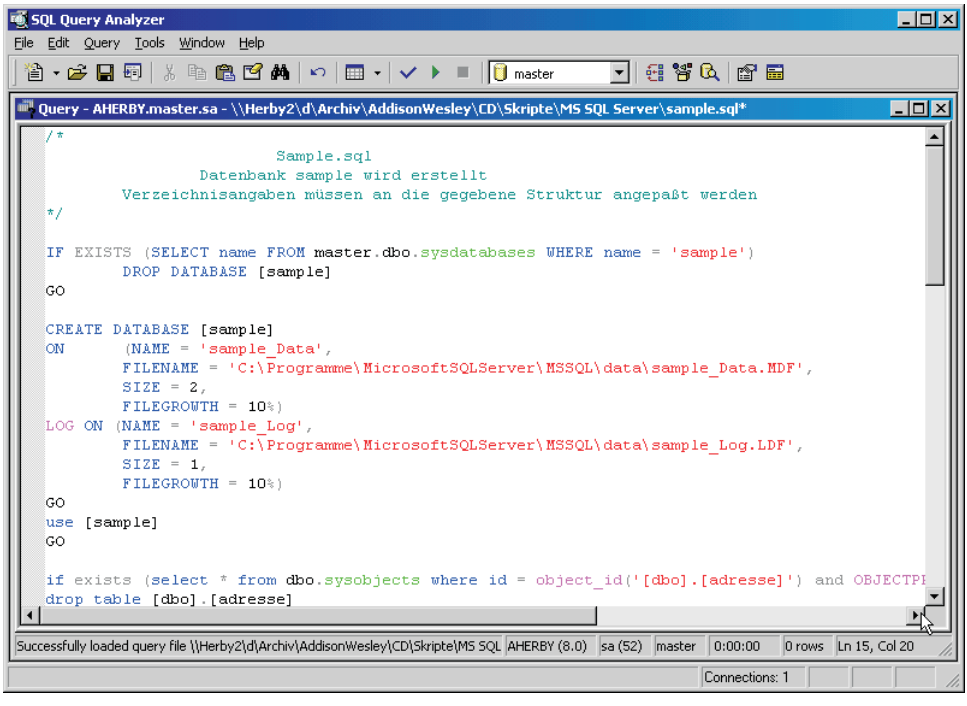


Abbildung 5.21: Ausführung eines Datenbankskriptes im Query Analyzer

5.8.3 Von der Datenbank zum SQL-Skript

Manche DBMS erlauben es, für die bestehenden Objekte im System die entsprechenden SQL-Anweisung zur Erstellung dieser Objekte als Skript zu generieren. Sie können also auch die Datenbank mit Unterstützung der grafischen Oberfläche entwerfen und nach der Fertigstellung ein Skript erzeugen, welches sie für einen späteren Neuaufbau verwenden können. Der SQL-Server bietet die Funktionalität, die das Erstellen von SQL-Skripten auf der Basis von vorhandenen Objekten möglich macht. Die notwendigen Arbeitsschritte werden in diesem Abschnitt vorgeführt.

1. Erweitern der Konsolenstruktur, bis die betreffende Datenbank erscheint.
2. Klicken Sie mit der rechten Maustaste auf die Datenbank.
3. Über ALLE TASKS im Kontextmenü wählen Sie SQL-Skript generieren.
4. Wählen Sie auf der Registerkarte ALLGEMEIN (General) die Datenbankobjekte aus, die in das Skript aufgenommen werden sollen (siehe Abbildung 5.22).

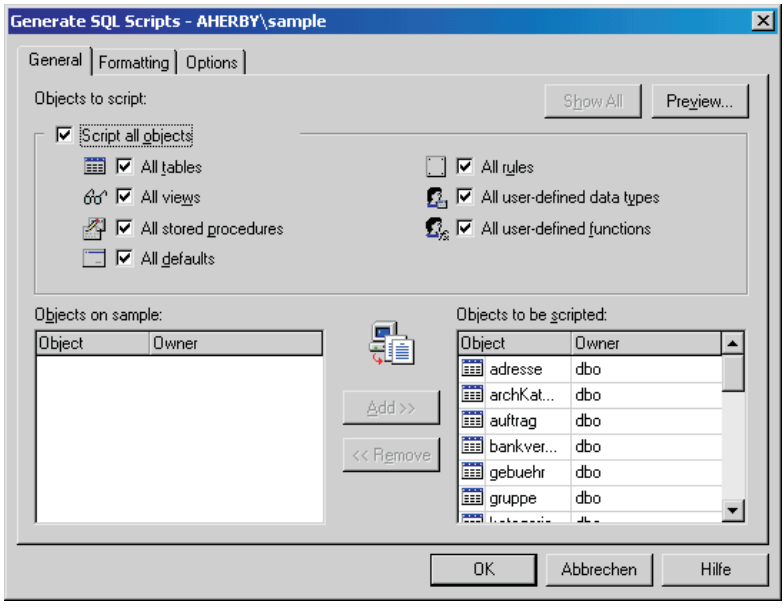


Abbildung 5.22: Erzeugen eines Skriptes im MS SQL Server

5. Auf der Registerkarte FORMATIERUNG können spezielle Formatierungsvorgaben definiert werden.

6. Trigger, Einschränkungen, Schlüssel etc. können auf der Registerkarte OPTIONEN zusätzlich für das Skript ausgewählt werden. Außerdem können Sie alle SQL-Anweisungen in einer Datei erstellen oder für jedes Objekt getrennt.
7. Geben Sie Namen und Verzeichnis für die Skriptdatei an.
8. Klicken Sie auf OK, um die Generierung zu starten.

Das Skript wurde in dem angegebenen Verzeichnis unter dem vorgegebenen Namen gespeichert. Das Skript kann nun in den Query Analyzer geladen und ausgeführt werden. Sie können das Skript auch in einem Texteditor laden und für ein anderes DBMS überarbeiten.

6 Eingabemasken

Für Zugriffe oder einfache Auswertungen z.B. des Controllers auf die Daten per SQL oder die Eingabe neuer Daten über eine Maske durch Mitarbeiter muss der einfache Zugriff auf die Daten hergestellt werden. Dieses Kapitel beschreibt neben den allgemeingültigen Grundlagen, einige Verfahren, die Sie beim Erstellen der Eingabemasken anwenden können. Es gibt Standardsoftware, die Sie ohne größere Anpassungen für die Dateneingabe verwenden können. Diese Programme werden ebenso vorgestellt, wie das Design und Programmieren einer neuen Oberfläche.

6.1 Die Eingabemaske – Schnittstelle zwischen Mensch und Maschine

Psychologen und Personalberater weisen immer wieder daraufhin, dass der erste Eindruck bei einem Vorstellungsgespräch besonders prägend ist. Dem Gesicht und der äußeren Erscheinung des Bewerbers kommen dabei besondere Bedeutung zu. Die Eingabemaske bzw. die graphische Oberfläche der Datenbankanwendung ist das Gesicht der Software. Die Fehler, die Sie bei der Gestaltung der *GUI* (Graphical User Interface) gemacht haben, können Sie durch starke Leistungswerte selten wieder gutmachen. Schlechtes GUI-Design macht sich nachteilig bei der Akzeptanz der Anwender bemerkbar und wird dadurch zu einem wichtigen Kriterium bei Kaufentscheidungen.

Dem Softwareentwickler muss klar sein, dass den Anwender bei weitem nicht die Technikverliebtheit auszeichnet, wie es bei ihm selbst der Fall ist. Der Anwender ist selten bereit länger als fünf Minuten nach der gesuchten Funktionalität zu suchen. Für den Anwender ist die Software ein Mittel zum Zweck und wenn die Anwendung den Zweck nur schlecht oder gar nicht erfüllt, dann wird er nach Alternativen suchen. Zwar muss man heute nicht mehr mit einer grundsätzlich negativen Einstellung der Anwender bei der Arbeit mit EDV-Systemen rechnen, jedoch werden neue Systeme mit großem Argwohn bedacht. Die Software muss dem Benutzer die Arbeit erleichtern. Dazu gehört, dass beim GUI-Design neben der Erfüllung funktionaler Rahmenbedingungen auch ergonomische und ästhetische Herausforderungen bedacht werden.

Man mag sich als Entwickler vielleicht auf den Standpunkt stellen, der Anwender wird sich schon an die neue Benutzeroberfläche gewöhnen und dies sei halt ein weiterer evolutionärer Schritt. Mit Produkten, die unter solchen Voraussetzungen entstanden sind, wird man kaum offene Türen einrennen. Wer nicht die menschlichen Eigenschaften und insbesondere seinen Arbeitsgewohnheiten berücksichtigt, begeht einen Kardinalfehler.

Beispiel 6.1:

Beim Besuch in einem Schwimmbad konnte ich mich von mangelnder Akzeptanz von neu eingeführter Technik überzeugen. Es gab ein Kassenhaus, welches mit einer Kassiererin besetzt war und zwei Kassenautomaten. An dem Kassenhaus bildete sich eine Schlange von ca. 30 Personen. Vor den beiden Automaten standen jeweils nur ein halbes Dutzend Badewilliger. Diejenigen, die ihr Ticket am Automaten erwarben, konnten direkt die Drehtür passieren und ihrer Badelust frönen. Meine Verwunderung war dementsprechend groß, dass die meisten Leute sich lieber am Kassenhaus anstellten und dabei die längere Wartezeit in Kauf nahmen. Natürlich versuchte ich das Ticket für mich und meine Frau vom Automaten zu holen. Der erste Automat war defekt und war deshalb nicht sonderlich hilfreich. Der zweite Automat war offensichtlich funktionstüchtig, denn es dauerte doch recht lang, bis sich die Personen, die unmittelbar vor dem Automaten standen von ihm losreißen konnten. Aber wenn sie sich vom Automaten entfernten gingen sie mit einem Ticket. Das ließ hoffen. Allerdings dauerte es ziemlich lang bis man sich in der vergleichsweise kurzen Menschenschlange vorwärts bewegte. Als ich dann selbst vor dem Automaten stand wusste ich warum. Man musste sich durch eine recht umständliche Verflechtung von Dialogfeldern auf dem Display navigieren, bei der es auch Sackgassen gab. Wurde eine falsche Badezeit eingegeben, so musste man mehrere Schritte zurückgehen, was einige Zeit in Anspruch nahm. Besonders ärgerlich war jedoch, dass jeweils nur ein Ticket ausgegeben wurde. Für meine Frau musste ich die gleiche umständliche Prozedur erneut ausführen. So war klar, dass viele Badegäste lieber bei der Kassiererin die Tickets kauften, weil diese Fragen beantworten konnte, genug Wechselgeld zur Verfügung hatte, mehrere Tickets gleichzeitig verkaufen konnte und dazu ein charmantes Lächeln für die Gäste übrig hatte.

Das »Gesicht« spielt eine große Rolle, wenn es darum geht, ob eine neue Technologie angenommen wird oder nicht.

6.1.1 Wahrnehmung beim Menschen

Auch wenn es nicht direkt mit der Datenbankprogrammierung im Zusammenhang steht, sei ein kurzer Exkurs in den Wahrnehmungsprozess beim Menschen gestattet. Der Ablauf beim menschlichen Wahrnehmungsprozess lässt Rückschlüsse zu, welche Qualitätsmerkmale eine Benutzeroberfläche haben sollte. Bei der Wahrnehmung geht

es darum, den ungeordneten Input, den der Mensch über die Sinnesorgane erfährt, auszuwerten.

Den Prozess der Wahrnehmung kann man in drei Abschnitte unterteilen: *sensorische Empfindung*, Wahrnehmung im engeren Sinn und *Klassifikation*.

Abbildung 6.1: Die drei Stufen der menschlichen Wahrnehmung

Die Abbildung 6.1 stellt diese drei Stufen dar. Bei der sensorischen Empfindung wird physikalische Energie (Licht oder Schallwellen) in neurale Energie umgeformt, d.h. die Gehirnzellen bekommen bestimmte Informationen über die Art des Reizes. Anhand der Merkmale, die durch Stimulation unterschiedlicher Rezeptorgane den Gehirnzellen übergeben werden, wird beim eigentlichen Wahrnehmungsprozess, ein passendes Muster bestimmt. Übergeordnete Gehirnprozesse werten also die Informationen aus, um sie bei der Klassifikation einer bekannten Kategorie zuzuordnen. Ein als geometrische Figur wahrgenommenes Objekt, wird bei der Klassifikation zum Fußball, Uhr oder Geldmünze, weil er die entsprechenden Eigenschaften aufweist.

Zwischen Wahrnehmung und Klassifikation lassen sich keine eindeutigen Grenzen ziehen. Bei der Wahrnehmung im Alltag sind sie eng ineinander verwoben und es sind keine eindeutigen Übergänge bemerkbar. Konzeptuell unterscheidet sich die Klassifikation allerdings von der Wahrnehmung dahingehend, dass sie auf bereits in der Vergangenheit erworbenes Wissen aufbaut und ein innerer Prozess höherer Ordnung ist. Die Wahrnehmung dagegen ist eher eine Kombination aus sensorischen Informationen und Klassifikationen.

Bei der visuellen Wahrnehmung wird beispielsweise der Betrachter einer Skyline durch das Netzhautbild zweidimensionale Objekte, wie Rechtecke, Quadrate und Trapeze zur Interpretation an das Gehirn weitergeben. Erst aufgrund von vorhandenem Wissen durch Vererbung oder Erfahrung, kann aus dem zweidimensionalen Netzhautbild ein Eindruck über die Objekte der realen Welt gewonnen werden.

Um die Millionen von Informationen, die wir über die Rezeptoren erhalten, als Gesamtbild interpretieren können findet eine Organisation der Wahrnehmung statt. Zu diesen Organisationsprozessen gehört die Gliederung in Bereiche, die Unterscheidung von Figur und Grund, Geschlossenheit der Objekte, Gruppierung der Objekte, Bezugsrahmen und das Prinzip der guten Gestalt.

Durch die Organisation der Wahrnehmung entstehen Gesamteindrücke. Es werden so nicht mehr nur Farbflecken oder einzelne Pixel wahrgenommen, sondern zueinander in Beziehung stehende Objekte.

6.1.2 Prinzip der guten Gestalt

Ist eine gegebene Region einmal abgegrenzt, als Figur vor dem Grund ausgewählt und zusammen mit ähnlichen Figuren gruppiert worden, so müssen die Grenzen noch zu bestimmten Formen organisiert werden. Die Organisationsprozesse der Formwahrnehmung sind empfindlich für etwas, das als Prinzip der guten Gestalt bezeichnet wird. Es schließt wahrgenommene Einfachheit, Symmetrie und Regelmäßigkeit ein.

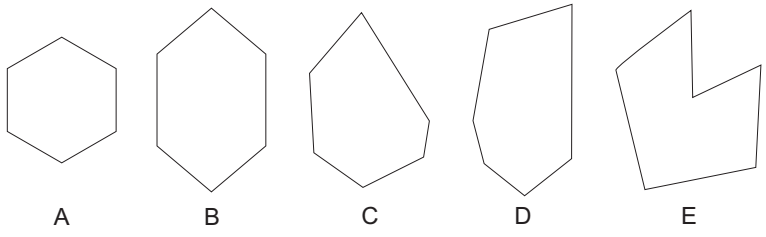


Abbildung 6.2: Prinzip der menschlichen Gestalt

Wenn Sie die Abbildung 6.2 betrachten, werden Sie wahrscheinlich die Figur A als die beste und die Figur E als schlechteste Gestalt empfinden. Experimente haben gezeigt, dass »gute« Gestalten leichter und genauer wahrgenommen werden. Man erinnert sich besser an sie und kann sie auch leichter beschreiben als die »schlechten« Gestalten. Als Ergebnis kann man formulieren, dass Formen »guter« Gestalt vom visuellen System effizienter kodiert werden können.

6.1.3 Gestaltungsgesetze

Aus den zuvor beschriebenen Kenntnissen der Wahrnehmungstheorie lassen sich als Verallgemeinerung einige Gestaltungsgesetze ableiten, die auch später in die Richtlinien für das GUI-Design einfließen werden:

1. Gesetz der Nähe:
   Schließen Sie Elemente zu einem Objekt zusammen, deren Distanz am geringsten ist.

2. Gesetz der Ähnlichkeit / der Gleichheit:
   Die Tendenz zum Zusammenschluss steigt mit wachsender Ähnlichkeit an.

3. Gesetz der guten Fortsetzung:
   Lassen Sie bei schneidenden Konturen die Linien möglichst wenig geknickt oder gebogen erscheinen.
4. Gesetz der Geschlossenheit:
   Konturen werden geschlossen wahrgenommen (Inneres zur Figur, Äußeres zum Grund).
5. Gesetz der Symmetrie:
   Zwischenräume zwischen symmetrischen Konturen werden eher zur Figur, als zwischen asymmetrischen Konturen.
6. Gesetz der Erfahrung:
   Erkennen von Oberflächen und ihren Grenzen muss von der Wahrnehmung geleistet werden (Infos über Beleuchtungs- und Abbildungsverhältnisse fehlen auf dem Netzhautbild).

6.2 Richtlinien für Funktionalität und Design

Mit der steigenden Verbreitung von EDV-Systemen und Standardsoftware wuchs auch das Bedürfnis nach Software mit anwenderfreundlichen Benutzeroberflächen. Es reichte nicht mehr aus, dass die Funktionalität allein der Maßstab für eine gute Software war. Die Systeme mussten auf den Arbeitsalltag der »breiten Masse« zugeschnitten werden. Die Bedienbarkeit der Programme wurde zu einem wichtigen Faktor bei der Entwicklung.

Für das Entwickeln von Software gibt es Normen. Dies haben wir bereits in Kapitel 2 gelernt. Insbesondere für die Benutzeroberfläche gibt es in der DIN 9241-10 einige ergonomische Anforderungen für »Bürotätigkeiten mit Bildschirmgeräten«.

6.2.1 Aufgabenangemessenheit

Informationsgehalt und Informationsfluss sollten an die durchzuführende Aufgabe angepasst werden. Erschlagen Sie den Anwender nicht mit einer Informationsflut, die er nicht überschauen kann. Gliedern Sie die Information nach hierarchischem System, so dass die für die bevorstehende Aufgabe wichtigste Information zuerst angezeigt wird und Hinweise mit geringer Priorität erst bei Interaktion durch den Benutzer eingeblendet wird.

Gestalten Sie den Weg, den der Benutzer zu gehen hat, um an seine Information zu gelangen, so einfach wie möglich. Je mehr Masken und Dialogfelder geöffnet und auch manuell wieder geschlossen werden müssen, desto mehr Aktion ist vom Anwender gefordert. Führen Sie den Benutzer durch »intelligente« Auswertung seiner Aktionen

und führen Sie ihn nicht in »Sackgassen«, aus denen er nur schwer wieder herauskommt. Setzen Sie den Fokus auf das Steuerelement, welches der Anwender auch im nächsten Schritt logischerweise verwenden würde.

Verzichten Sie auf unnötige Spielereien, die den Anwender nicht direkt zum Ziel bringen. Ein Abrechnungssystem für deutsche Steuerberater braucht mit Sicherheit keinen Euro-Yen-Umrechner. Bei der Softwareentwicklung ist vieles möglich. Nicht alles, was möglich ist, sollte in die Software implementiert werden.

6.2.2 Selbstbeschreibungsfähigkeit

Selten haben die Anwender Zeit, sich wochenlang in die neue Software einzuarbeiten. Eine wichtige Forderung ist deshalb die *Selbstbeschreibungsfähigkeit* der Software. Im Idealfall ist der Anwender bei jedem Schritt und bei jedem Vorgang über die Bedienbarkeit oder ggf. die Systemprozesse informiert, ohne dass er dafür zusätzlich geschult werden muss. Manche Funktionen sind so gut versteckt, dass sie selbst nach dem Studieren des Handbuches dem Anwender unerschlossen bleiben.

Abbildung 6.3: Beispiel für selbstbeschreibende Programmoberfläche

Geben Sie dem Benutzer Informationen über Tastenkombinationen, wenn solche implementiert sind. Setzen Sie den Benutzer über einen laufenden Vorgang über eine Angabe in der Statusleiste in Kenntnis. Geben Sie Benutzerfeedback, um den Anwender bei seinen Auswahlmöglichkeiten zu unterstützen. Abbildung 6.3 zeigt eine Benutzeroberfläche, die auf unaufdringliche Art und Weise den Benutzer über gewählte Voreinstellungen informiert und zusätzlich die Tastenkombination für die wichtigste Aktion anzeigt.

6.2.3 Steuerbarkeit

Eine Applikation ist steuerbar, wenn der Anwender den Dialogablauf starten und seine Richtung und Geschwindigkeit beeinflussen kann, bis das Ziel erreicht ist. Es sollte immer der Anwender sein, der Vorgänge in Gang setzt. Mit Menüs, Toolbars und Tastenkombinationen hat der Anwender drei Möglichkeiten, den Programmablauf zu steuern. Neben den Korrekturmöglichkeiten und Eingabevorschlägen durch das Programm, sollte es immer der Anwender selbst sein, der den Ablauf steuert. Dies setzt natürlich voraus, dass er zuvor über die einzelnen Schritte in Kenntnis gesetzt wurde oder die notwendigen Tätigkeiten selbsterklärend sind.

6.2.4 Erwartungskonformität

Es wurde beschrieben, dass der Mensch bei der Wahrnehmung Objekte nach bekannten Mustern klassifiziert. Formen oder Vorgänge, die er bereits kennt, kann er wesentlich schneller bearbeiten, als dies bei unbekannten der Fall ist. Der Benutzer eines Programms hat heute typischerweise gewisse Vorkenntnisse im Bereich der EDV-Anwendungen. Durch die weite Verbreitung von Windows-Applikationen hat sich deshalb auch so etwas wie ein Standard durchgesetzt, der sich an Windows-Anwendungen anlehnt. Jeder Benutzer erwartet von einer neuen Anwendung ein Verhalten, das ihm bereits aus anderen bekannt ist.

Beispiel 6.2:

Wenn Sie in einer Windows-Anwendung nach einem bestimmten Begriff suchen, dann können Sie normalerweise mit der Tastenkombination Ctrl und F ein Dialogfeld wie in Bild 6.4 gezeigt öffnen, in dem Sie den Suchbegriff und Suchrichtung bestimmen und anschließend den Suchvorgang starten können.

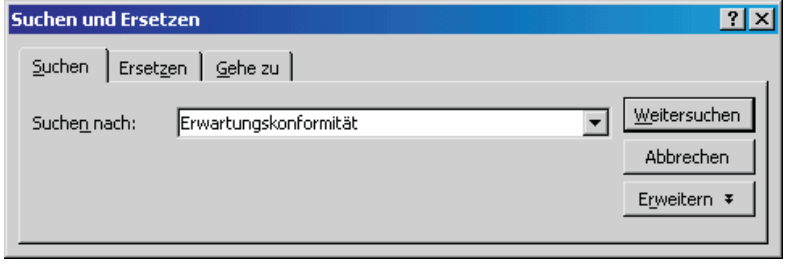


Abbildung 6.4: Suchfunktion bei Windows-Anwendungen

Auch die Tastenkombinationen zum Ausschneiden, Kopieren und Einfügen sollten von Ihnen nicht für andere Zwecke missbraucht werden. Für die gesamte Benutzeroberfläche gibt es gewisse Standards, die teilweise durch die verwendeten Steuerele-

mente und Standarddialoge vorgegeben sind. Nutzen Sie die Standarddialoge (z.B. Dateisuche, Drucken, Speichern unter etc.), wenn Sie in der Funktionalität ihren Anforderungen entsprechen. Wenn es nicht unbedingt erforderlich ist, sollten Sie nicht eigene Steuerelement kreieren. Das kostet nicht nur wertvolle Entwicklungszeit, sondern behindert den Wiedererkennungsprozess des Benutzers. Die Erwartungskonformität ist ein wichtiger Aspekt, wenn es darum geht, ob die Software beim Anwender auf Zustimmung stößt.

Allerdings ist es nicht erforderlich alles blind zu übernehmen. Wenn Sie Mängel bei bestehenden Standards erkannt haben, dann können Sie diese natürlich durch optimierte Elemente ersetzen.

6.2.5 Fehlertoleranz

Innerhalb der Testphase müssen Sie die Applikation unbedingt auf ihre Fehlertoleranz prüfen. Die Applikation muss gegenüber falschen Eingaben und Aktionen eine Robustheit aufweisen. Es darf nicht passieren, dass ein Programm aufgrund einer falschen Benutzeraktion nicht mehr steuerbar ist.

Außerdem wird dem Anwender die Angst vor möglichen Fehlern genommen, wenn ihm bewusst ist, dass er seine Aktionen rückgängig machen kann. Funktionen wie ZURÜCK, ABBRUCH oder REGENERIEREN beruhigen nicht nur den Einsteiger. Implementieren Sie Funktionalität, die falsche Benutzeraktionen rückgängig machen und informieren Sie ausreichend über die vorhandenen Möglichkeiten. Insbesondere bei komplexen Datenbankabfragen steht der Benutzer ständig in der Versuchung, den Vorgang abzubrechen. Wenn ihm das Programm nicht die Möglichkeit gibt, sucht er sich einen anderen Weg. Beim Abbruch einer Datenbankaktion muss darauf geachtet werden, dass die Datenbankverbindung und eventuell angelegte temporäre Ressourcen wieder abgebaut werden.

Erstellen Sie Software, die intelligent genug ist, um Fehler durch den Anwender präventiv zu verhindern. Deaktivieren Sie Steuerelemente, die im Augenblick nicht relevant sind oder unter der gegebenen Konstellation zu Fehlern führen. Statten Sie Ihre Software mit Fehlerkorrektur aus. Gibt der Benutzer z.B. in einem Datumsfeld »020202« an, dann könnte eine entsprechende Funktion den Wert umformen, so dass er den Integritätsregeln der Datenbank entspricht und dort gespeichert werden kann.

6.2.6 Individualisierbarkeit

Dem Benutzer sollte die Möglichkeit eingeräumt werden, dass er Einstellungen vornimmt, die sowohl die Benutzeroberfläche, als auch die Reihenfolge von Arbeitsschritten verändert. Er soll die Software auf seine individuellen Bedürfnisse anpassen können. Das Bedürfnis nach Anpassungsfähigkeit der Software hängt natürlich in hohem Maße auch von den Erfahrungen des Benutzers ab. Für den Einsteiger ist es

sicherlich von Vorteil, wenn die Software mit ihrem Optionen überschaubar ist. Der »alte Hase« dagegen erwartet, dass er Einstellungen vornehmen kann, die er bereits aus anderen Anwendungen kennt.

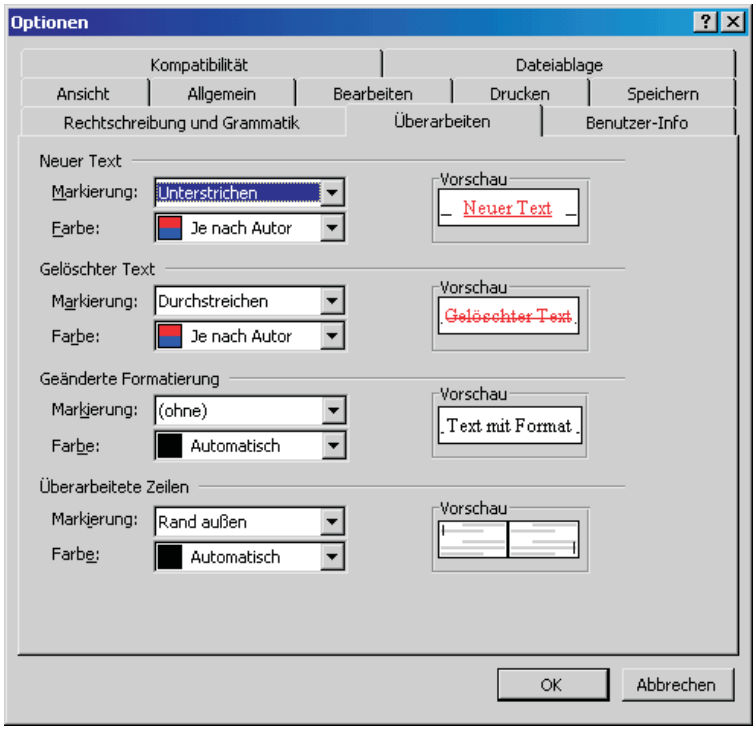


Abbildung 6.5: Benutzereinstellungen in MS Word

In MS Word können Sie z.B. über OPTIONEN eine Vielzahl von Einstellungen vornehmen, die sich auf Layout und Funktionalität gleichermaßen auswirken. Zum Standard gehört es mittlerweile auch, dass zuvor gewählte Angaben von Verzeichnissen beim nächsten Aufruf des entsprechenden Dialogfeldes vom Programm gemerkt wurden und dadurch ein mühsames Klicken durch den Verzeichnisbaum erspart bleibt.

Eine Erleichterung kann auch das Zusammenfassen von mehreren Arbeitsschritten sein, die der Benutzer bei Bedarf durch Klicken auf ein Icon aktivieren kann.

Die benutzerdefinierten Einstellungen können Sie in die Benutzerdatenbank selbst, eine Ini-Datei oder in die Registrierungsdatenbank schreiben.

6.3 Steuerelemente für die Eingabemaske

Wenn wir zuvor die Benutzeroberfläche mit einem Gesicht verglichen haben, dann können wir dieses Bild weiterhin gebrauchen und die Steuerelemente (Control) als Augen oder Mund verstehen. Durch die Steuerelemente ist es möglich, Daten auf Eingabemasken oder in Berichten darzustellen. Der Benutzer gibt z.B. einen Zahlenwert über ein Textfeld auf der Eingabemaske (oder auch Formular) ein. Dieser Zahlenwert kann später mit einem Bericht abgerufen werden. Auch hier ist es wieder das Steuerelement Textfeld, welches den Wert auf dem Bericht zur Anzeige bringt. Eingabe und Ausgabe der Daten können mithilfe von Steuerelementen durchgeführt werden. Es hängt von der Entwicklungsumgebung ab, welche Steuerelemente dem Programmierer zur Verfügung stehen. Die Steuerelemente, die in diesem Abschnitt vorgestellt werden, gehören aber mit Sicherheit zur Grundausstattung und sind für die Gestaltung einer Eingabemaske elementar.

Abbildung 6.6: Steuerelemente in der Entwicklungsumgebung

6.3.1 Bezeichnungsfelder

Ein Bezeichnungsfeld (Label) ist ein grafisches Steuerelement, mit dem Text angezeigt wird, den der Benutzer nicht direkt ändern kann. Sie können dieses Steuerelement irgendwo in Ihrem Bericht oder Formular positionieren, um Text auszugeben. Es besteht keine Verbindung zur Datenbank, so dass der auszugebende Text statisch ist.

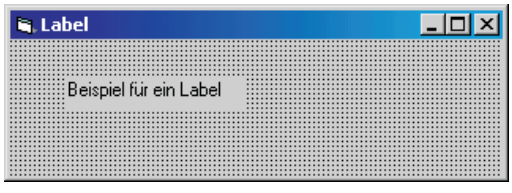


Abbildung 6.7: Das Steuerelement Bezeichnungsfeld

Abbildung 6.7 zeigt ein Formular, auf dem ein Bezeichnungsfeld positioniert wurde. Über die Eigenschaften des Steuerelementes können Beschriftung, Name, Schriftart, Schriftgröße, Position etc. geändert werden. Bezeichnungsfelder finden wir oftmals zusammen mit Textfeldern, die sie beschreiben.

6.3.2 Textfelder

In Textfelder (Textbox) können beliebige Texte eingegeben werden. Textfelder dienen sowohl der Ein- als auch der Ausgabe von Daten. Wenn Sie das Steuerelement jedoch mit einem Schreibschutz belegen, dann ist es dem Benutzer nicht möglich, Daten über dieses Textfeld einzugeben oder angezeigte Daten zu ändern. Sie können Textfelder auch über mehrere Zeilen hinweg eingeben, wobei der Zeilenumbruch mit der Tastenkombination Ctrl und Enter herbeigeführt wird. Bei den Eigenschaften ist die mehrzeilige Funktionalität einzustellen. Die Formatierungsmöglichkeiten innerhalb eines Textfeldes sind meistens nicht sehr komfortabel. Der Text kann üblicherweise nur in einer Schriftart, -auszeichnung und -größe dargestellt werden. Reicht dies nicht aus und stößt man auch durch die maximal erlaubte Textlänge auf die Grenzen des Textfeldes, so kann man auf erweiterte Steuerelemente wie die *Rich-Textbox* zurückgreifen.

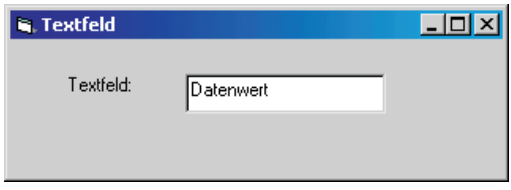


Abbildung 6.8: Das Steuerelement Textfeld

Die Abbildung 6.8 zeigt eine Modifizierung des Formulars aus Abbildung 6.7. Das Bezeichnungsfeld wurde mit einer anderen Beschriftung versehen und zusätzlich wurde ein Textfeld eingefügt. Wie in der Abbildung zu sehen ist, kann im Ausführungsmodus ein Datenwert in das Textfeld eingegeben werden. Wenn das Textfeld direkt mit einem Datenbankfeld verknüpft ist, dann wird der Datenwert auch direkt in die Datenbank geschrieben. Andernfalls kann eine Programmroutine veranlassen, dass alle Datenwerte aus den Textfeldern des Formulars ausgelesen und in die Datenbank geschrieben werden.

6.3.3 Kombinations- und Listenfelder

Die Steuerelemente *Listenfelder* (Listbox) und *Kombinationsfelder* (Combobox) bieten dem Benutzer eine Liste mit Auswahlmöglichkeiten an. Standardmäßig werden diese Auswahlmöglichkeiten in einer einspaltigen senkrechten Liste angezeigt. Sie können aber auch mehrspaltige Listen anlegen. Wenn die Anzahl der Elemente zu groß wird, um vollständig im Kombinations- oder Listenfeld angezeigt zu werden, werden dem Steuerelement automatisch Bildlaufleisten hinzugefügt. Der Anwender kann anschließend in der Liste von oben nach unten bzw. von links nach rechts blättern.

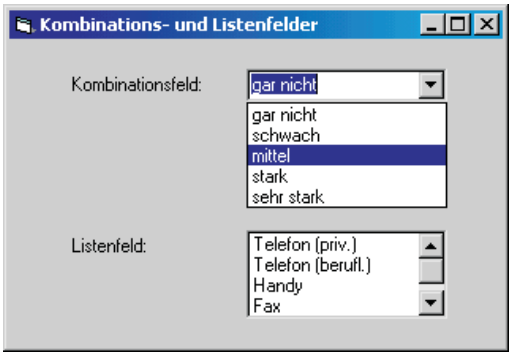


Abbildung 6.9: Kombinations- und Listenfelder

Das Formular aus Bild 6.9 enthält sowohl ein Kombinations- als auch ein Listenfeld. Neben den offensichtlichen äußeren Unterschieden der beiden Steuerelementen gibt es Unterschiede in der Auswahl der Einträge. Im Kombinationsfeld kann der Benutzer selbst Text eingeben oder er wählt einen Eintrag aus der Auswahlliste aus. Somit verbindet ein Kombinationsfeld die Möglichkeiten eines Textfeldes mit denen eines Listenfeldes. Die Einträge der Auswahllisten können bei beiden Steuerelementen auch aus der Datenbank gefüllt werden.

6.3.4 Optionsfelder

Soll von dem Benutzer eine von mehreren möglichen Optionen ausgewählt werden, so wird meistens das *Optionsfeld* eingesetzt. Das Steuerelement *Optionsfeld* zeigt eine Option an, die ein- oder ausgeschaltet werden kann.

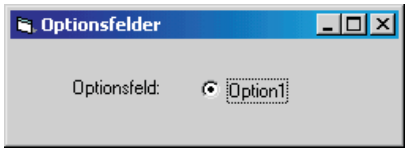


Abbildung 6.10: Das Steuerelement Optionsfeld

Optionsfelder werden meistens in einer Optionsgruppe zusammengefasst. Eine Optionsgruppe besitzt einen Ergebniswert, aus dem sich eine Aussage über die gewählte Option ableiten lässt. In der Abbildung 6.11 ist ein Anwendungsbeispiel einer Optionsgruppe dargestellt.

In einer Optionsgruppe kann jeweils nur ein Optionsfeld selektiert werden.

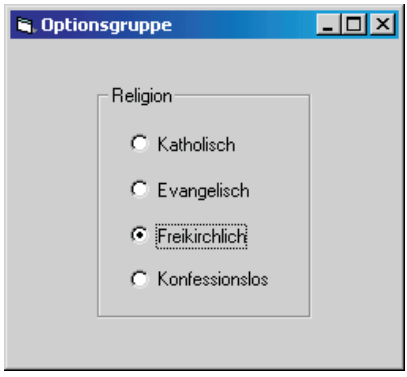


Abbildung 6.11: Beispiel für eine Optionsgruppe

6.3.5 Kontrollkästchen

Das *Kontollkästchen* (Checkbox) ist ein Steuerelement, welches angibt, ob eine Option aktiviert oder deaktiviert ist. Im Kontrollkästchen wird ein Häkchen angezeigt, wenn sie aktiviert ist (siehe Abbildung 6.12). Kontrollkästchen sind unabhängig voneinander, deshalb kann ein Benutzer beliebig viele Kontrollkästchen gleichzeitig aktivieren.

Abbildung 6.12: Das Steuerelement Kontrollkästchen

Sie können Kontrollkästchen, oder Gruppen von Kontrollkästchen, für Felder vom Typ *boolean* verwenden. Die Steuerelemente Kontrollkästchen und Optionsfelder haben ähnliche Eigenschaften. Im Gegensatz zur Gruppe von Optionsfeldern können bei gruppierten Kontrollkästchen jedoch mehrere gleichzeitig aktiviert sein.

6.3.6 Rahmen

Das *Rahmen-Steuerelement* (Frame) wird zur logischen und optischen Gruppierung von anderen Steuerelementen verwendet. So können beispielsweise mit einem Rahmen-Steuerelement eine Gruppe von Kontrollkästchen funktionell in einem Formular separiert werden (siehe Abbildung 6.13).

Abbildung 6.13: Das Rahmen-Steuerelement

In den meisten Fällen wird das Rahmen-Steuerelement passiv, zur Gruppierung von anderen Steuerelementen eingesetzt.

6.3.7 Befehlsschaltflächen

So wie Sie mit dem Lichtschalter das Licht ein oder aus schalten, können Sie mit dem Steuerelement *Befehlsschaltfläche* (CommandButton) Aktionen starten oder beenden. Befehlsschaltflächen bieten dem Anwender eine leicht erreichbare Schaltfunktion, um Prozesse zu aktivieren. Klickt der Benutzer auf eine Schaltfläche, so wird eine Prozedur aufgerufen und die Anweisungen der Prozedur ausgeführt.

Abbildung 6.14: Die Befehlsschaltfläche

Wenn der Benutzer die Schaltfläche wählt, wird nicht nur die entsprechende Operation durchgeführt, sondern die Schaltfläche sieht auch so aus, als ob sie niedergedrückt und losgelassen würde. In den meisten Fällen können die Befehlsschaltflächen auf zwei Wegen erreicht werden, über einen Mausklick und über Tastenkombinationen (siehe auch Abbildung 6.14).

6.3.8 Register

Wollen Sie mehrere Steuerelemente thematisch zusammenstellen ohne jeweils ein neues Fenster zu verwenden, dann können Sie mit dem Steuerelement *Register* arbeiten. Sie können ein Register-Steuerelement verwenden, um mehrere Seiten mit Informationen als Einheit zur Verfügung zu stellen. Dies ist besonders dann sinnvoll,

wenn Sie mit einer größeren Anzahl von Steuerelementen arbeiten, die sich in zwei oder mehr Kategorien einteilen lassen.

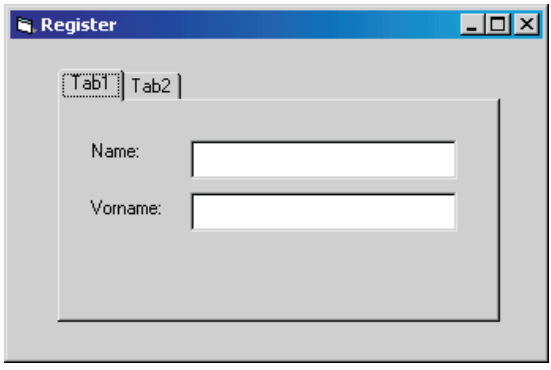


Abbildung 6.15: Das Register-Steuerelement

Wie in Abbildung 6.15 zu sehen, können Sie z.B. das Steuerelement *Register* dafür verwenden, die Eingabemaske nach Kategorien aufzuteilen. Die Zusammenhänge der Entitäten und ihren Attributen, die sich auch in der Tabellenstruktur widerspiegelt, kann hier auch auf die einzelnen Registerkarten abgebildet werden. Es ist auch möglich, mit Register-Steuerelementen in Dialogfeldern zu arbeiten, um so die Informationen zu bündeln. Über die Karteireiterfenster kann der Benutzer selbst die Reihenfolge seiner Dateneingabe bestimmen.

6.3.9 Dialogfelder

Dialogfelder dienen ebenfalls der Ein- und Ausgabe von Daten. Beim Anklicken einer Befehlsschaltfläche kann beispielsweise ein *Dialogfeld* geöffnet werden, in dem weitere Informationen oder Steuerelemente zur Interaktion enthalten sind. Das Dialogfeld kann auch als spezielle Art des Formulars angesehen werden. Es gibt eine Reihe von Standard-Dialogfeldern für die DRUCKERANSTEUERUNG, DATEISUCHE, DATEI ÖFFNEN etc (siehe Abbildung 6.16).

In Abbildung 6.16 wurde das Standard-Dialogfeld *Drucken* über eine Befehlsschaltfläche aufgerufen. Bei den Standard-Dialogfeldern ist es nicht mehr notwendig, sich über das Design Gedanken zu machen. Anders ist es bei den benutzerdefinierten Dialogfeldern, die Sie selbst erstellen. Im Folgenden seien die acht Regeln zum Dialogdesign nach *Shneiderman* aufgeführt.

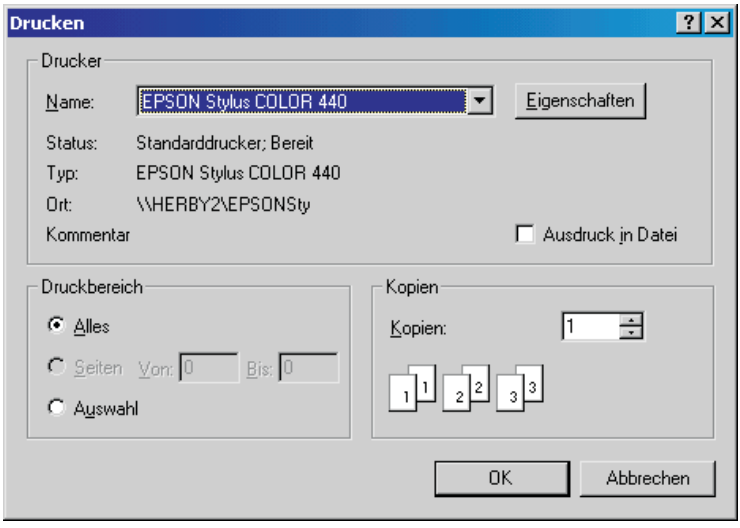


Abbildung 6.16: Das Standard-Dialogfeld Drucken

1. Versuchen Sie Konsistenz zu erreichen.
   - ähnliche Situationen, ähnliche Aktionsfolgen.
   - Verwendung identischer Begriffe.
2. Bieten Sie erfahrenen Benutzern Abkürzungen an.
   - Abk. von Kommandos, Fkt.-Tasten, Makros usw. sparen Zeit.
3. Bieten Sie informatives Feedback.
   - Visualisierung der Arbeitsobjekte.
   - Aktion führt zu sichtbarer Reaktion.
   - Umfang des Feedbacks orientiert an Tragweite der Aktion.
4. Dialoge sollten abgeschlossen sein.
   - Anfang und Ende klar erkennbar.
   - Menschen wollen Handlungen abschließen.
5. Bieten Sie einfache Fehlerbehandlung.
   - System muss schwerwiegende Fehler verhindern.
   - Bei Fehlern: Korrekturvorschläge.

6. Bieten Sie einfache Rücksetzmöglichkeiten.
   - Undo-Möglichkeit schafft Sicherheit und reduziert Angst.
7. Unterstützen Sie benutzergesteuerten Dialog.
   - Keine unerwarteten Systemreaktionen.
   - Problemloser Abruf von Informationen.
   - Einfache Ausführung von Aktionen.
8. Reduzieren Sie die Belastung des Kurzzeitgedächtnisses.
   - Einfache Bildschirminhalte.
   - Hilfemöglichkeit für Syntax, Abk. und Codes.

6.3.10 Fokus

Stellen Sie sich vor, Sie müssen täglich nahezu hundert neue Kunden über eine Eingabemaske in die Datenbank einfügen. Für jeden Kunden gibt es mehrere Eigenschaften, d.h. Sie haben jeweils für die Kunden vielleicht 30 oder 40 Felder zu füllen. Natürlich versuchen Sie Ihre Dateneingabe so zu optimieren, so dass Sie so schnell wie möglich die Daten eingeben können. Sie werden feststellen, dass Sie wesentlich schneller vorankommen, wenn Sie ausschließlich mit der Tastatur arbeiten und ⇆ verwenden, um zum nächsten Eingabefeld zu springen. Dies ist aber nur dann eine Arbeitserleichterung, wenn Sie mit ⇆ tatsächlich zu dem Feld kommen, welches in einer logischen Reihenfolge das nächste wäre. Die Steuerelemente verfügen über Fokus-Eigenschaften. Der Benutzer kann nur über das Steuerelement, welches den aktuellen Fokus hat, Eingaben machen. Auf der Windows-Oberfläche können mehrere Anwendungen gleichzeitig laufen, aber nur die Anwendung, die den Fokus hat, verfügt über eine aktive Titelleiste und kann Eingaben des Benutzers empfangen.

Die Reihenfolge, die durch Betätigen der ⇆ auf einer Eingabemaske durchlaufen wird, nennt man auch *Aktivierreihenfolge*. Jedes Formular besitzt seine eigene Aktivierreihenfolge. Normalerweise stimmt die Aktivierreihenfolge mit der Reihenfolge überein, in der die Steuerelemente erstellt wurden. Sie können die Reihenfolge über die Eigenschaften der Steuerelement verändern. Der Fokus kann auch zur Laufzeit des Programmes in Abhängigkeit von bestimmten Ereignissen dynamisch gesetzt werden (siehe Abbildung 6.17).

Beim Öffnen eines Fensters ist darauf zu achten, dass das jeweils erste Steuerelement den Fokus erhält und die Aktivierreihenfolge der logischen Bearbeitungsreihenfolge entspricht. Im Bild 6.17 sehen Sie auch, dass der Fokus nach dem letzten Eingabefeld auf eine Befehlsschaltfläche übergehen sollte, welche durch Anklicken die eingegebenen Werte sichert und die aktuelle Maske verlässt.

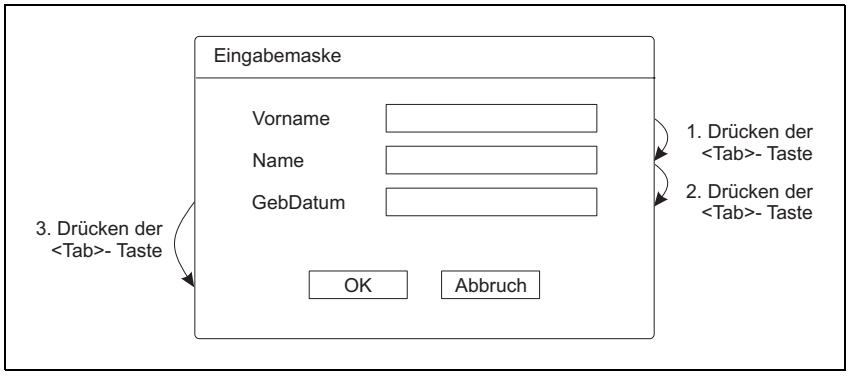


Abbildung 6.17: Beispiel für Aktivierreihenfolge

6.4 Vermeidbare Fehler beim GUI-Design

Wenn Sie mit Ihrer Benutzeroberfläche nicht einen Preis für das schlechteste Design und unzureichende Funktionalität gewinnen wollen, sollten Sie zumindest einige grundsätzliche Aspekte beim Entwurf beachten. Es gibt einige Dinge, die macht man einfach gefühlsmäßig richtig, ohne sich längere Zeit darüber Gedanken machen zu müssen. Bei der Verwendung von ausgereiften Entwicklungsumgebungen gibt es viele Standardeinstellungen, welche die ganz großen Vergehen beim Design gar nicht zulassen. Trotzdem sollten Sie nach Erstellung Ihrer Benutzeroberfläche nochmals überprüfen, ob Sie nicht doch einen der hier genannten Fehler begangen haben.

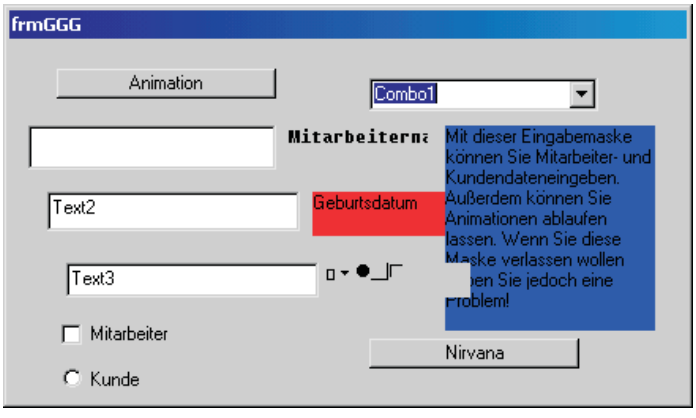


Abbildung 6.18: Dialogfeld mit vermeidbaren Fehlern

6.4.1 Falsche Farbenwahl

Mit Farben können Sie Gruppierungen vornehmen; seien Sie jedoch vorsichtig bei der Auswahl der Farben. Verwenden Sie nie Rot und Blau nebeneinander, denn das Auge kann nicht beides scharf abgleichen. Bestimmte Farben werden auch mit gewissen Bedeutungen verbunden. Je nach Kulturkreis gelten Schwarz und Weiß als Farben der Trauer. Die Farbe Rot wird oftmals mit Warnhinweisen und Grün mit Entwarnungshinweisen in Zusammenhang gebracht.

Verwenden Sie im Allgemeinen nicht zu viel verschiedene Farben und wählen Sie für den Hintergrund diskrete Farben.

6.4.2 Zu viele Elemente in einem Fenster

Die Steuerelemente in einem Fenster und insbesondere in einem Dialogfeld müssen überschaubar sein. Der Benutzer wird es sicherlich nicht schätzen, wenn Sie beim Aufteilen Ihrer Steuerelemente mit dem zur Verfügung stehenden Platz extrem sparsam sind. Ein gewisser Leerraum im Fenster ist durchaus sinnvoll und hilft dem Anwender bei der Orientierung. Bringen Sie Ihre Steuerelemente wenn nötig auf mehreren Fenstern unter. Ein hervorragendes Mittel bei der Aufteilung der Steuerelemente mit gleichem Informationshintergrund, bietet sich mit dem *Register-Steuerelement*.

Verwenden Sie auch nicht zu viel Text bei Bezeichnungsfeldern bzw. in Dialogen. Sie sollten nie mehr als 200 Zeichen für eine Benutzermeldung verwenden. Seien Sie bei der Wahl Ihrer Wörter möglichst präzise.

6.4.3 Keine Gruppierung verwendet

Das Dialogfeld in Abbildung 6.18 wirkt auch deshalb so unruhig und unübersichtlich, weil keine Gruppierung der Steuerelemente vorgenommen wurde. Steuerelemente, die inhaltlich zusammengehören, sollten auch direkt als Gruppe erkannt werden. Sie können dies mit Steuerelementen wie Rahmen, Register und auch Linien bewirken oder durch entsprechende Leerräume einen Abstand zwischen den Gruppen schaffen. Passen Sie auch die Höhe und Breite der Steuerelemente, sowie ihre Positionen innerhalb der Gruppen an, so dass die jeweiligen Gruppen kompakt und strukturiert erscheinen.

6.4.4 Verwendung exotischer Schriftarten

Versuchen Sie nicht, den inhaltlich schwachen Text mit einer besonders gelungenen Kombination der Schriftarten aufzuwerten. Damit der Anwender auch wirklich alle Texte lesen kann, sollten Sie sich auf Standardschriftarten (MS San Serif, Courier oder Arial) beschränken. Bei der Verwendung exotischer Schriftarten, kann es passieren,

dass Text wie in Abbildung 6.18 ausgegeben werden, weil auf dem PC des Anwenders die notwendige Schriftart nicht installiert ist. Wahllose Kombination der Schriftgrößen führt ebenso wie das Verwenden von fetten und kursiven Zeichen zu einem unruhigen Erscheinungsbild.

6.4.5 Schlecht angepasste Steuerelemente

Ein Textfeld, welches für die Ein- und Ausgabe eines Feldes verwendet wird, hat eine Breite von 50 mm. In der Datenbank hat das Feld nur eine Größe von drei Zeichen. Dies ist ein typisches Beispiel für Platzverschwendung auf der Eingabemaske. Dem Anwender wird suggeriert, er hätte hier mehr Zeichen einzugeben, als dies tatsächlich möglich ist. Das andere Extrem wäre ein Steuerelement, welches die tatsächlichen Werte der Datenbank nicht vollständig abbilden kann, weil die Größe nicht ausreicht. Bei der Wahl der zu verwendenden Steuerelementen, sowie ihrer Größe, ist stets der zu erwartende Datenwert zu berücksichtigen.

6.4.6 Falsche Aktivierreihenfolge

Im Abschnitt »Fokus« wurde bereits darauf hingewiesen, wie wichtig eine logische Reihenfolge beim *Fokussieren* ist. Geben Sie den Steuerelementen eine Aktivierreihenfolge, die an sinnvollen Arbeitsschritten angepasst ist. Setzen Sie nach dem letzten Eingabefeld nie den Fokus direkt auf eine Befehlsschaltfläche, welche einen Löschvorgang oder ein Schließen des Fensters ohne Speicherung der Daten bewirkt.

6.4.7 Den Benutzer in die Falle locken

Der Benutzer, der das Dialogfeld aus Abbildung 6.18 aufruft hat ein Problem, er kann dies Fenster nicht ohne Weiteres durch Klicken auf eine Befehlsschaltfläche schließen. Auf diesem Dialogfeld fehlt jegliche Funktionalität zum Schließen. Bringen Sie den Anwender nicht in Situationen, die er nicht ohne größere Anstrengungen bewältigen kann. Aus der Anordnung der Steuerelemente muss jeder durchzuführende Schritt selbsterklärend sein oder durch Hilfsdialoge erklärt werden. Beim Arbeiten mit Eingabemasken sollte es immer ein *Vor* und ein *Zurück* geben.

6.5 Standard-Applikationen, die sich als Frontend eignen

Nicht immer ist es erforderlich, die Benutzeroberfläche mit viel Programmieraufwand selbst zu erstellen. Manchmal reicht es schon aus, eine Schnittstelle zu schaffen, die es erlaubt, die Daten zu durchsuchen, ohne dass dem Anwender tiefere Kenntnisse von SQL abverlangt werden. Vielleicht ist auch schon eine Anwendung vorhanden, an die sich der Anwender gewöhnt hat und mit der er auf die Datenbank zugreifen will. Für

diese Fälle müssen Sie innerhalb der Standardanwendungen nach Schnittstellen für Ihre Datenbank suchen. Im Zusammenhang mit dieser Problemstellung sollen drei Applikationen im Zusammenspiel mit der Datenbank beschrieben werden.

6.5.1 Mit Excel auf die Datenbank zugreifen

In diesem Abschnitt wird gezeigt, wie Sie mit MS Excel auf die Daten Ihrer Datenbank zugreifen. Mit Excel können Sie eine Abfrage auf verschiedene Datenquellen durchführen. Sie könnten u.a. aus den Tabellen *kategorie* und *gebuehr* der Beispieldatenbank eine Preisliste für Ihre Dienstleistungen zusammenstellen und in Excel ausgeben (siehe Abbildung 6.19).

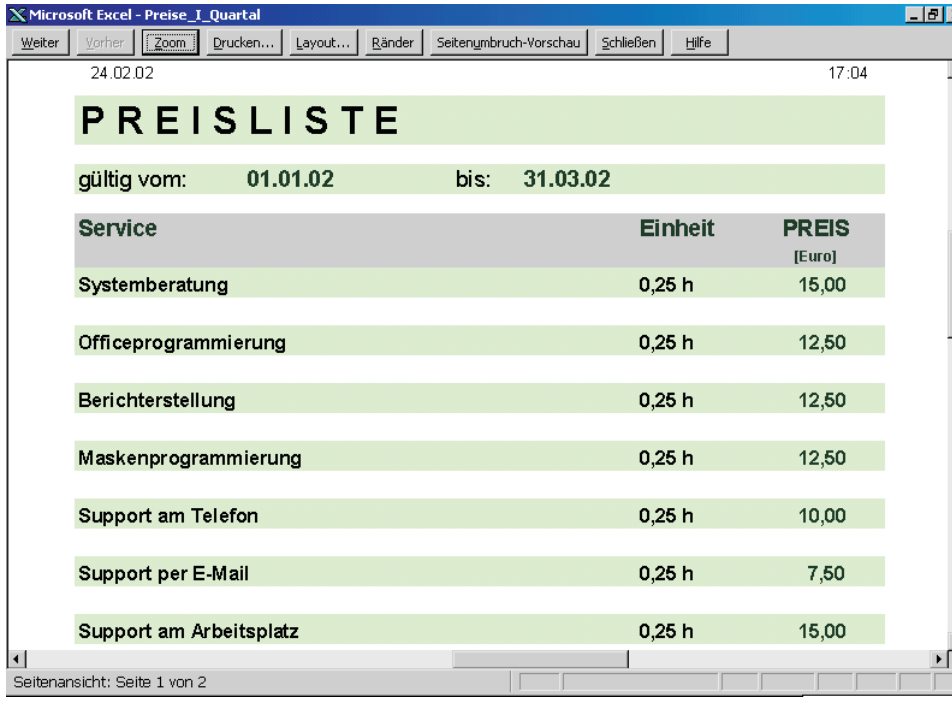


Abbildung 6.19: Listenausgabe in Excel

Das Modul MS Query, welches Sie in Excel optional installieren können, ermöglicht eine Verbindung zur Datenquelle. Mit einem so genannten Query-Assistenten erstellen Sie eine Abfrage für die bestimmte Datenquelle (DATEN/EXTERNE DATEN/NEUE ABFRAGE ERSTELLEN). Die Daten werden dann über die erstellte Abfrage tabellarisch, als Diagramm oder mit einer Maske in Excel verarbeitet. Sie können die Eingabemaske, DATEN/MASKE... aufrufen und die Daten dort durchblättern bzw. ändern (siehe Abbildung 6.20).

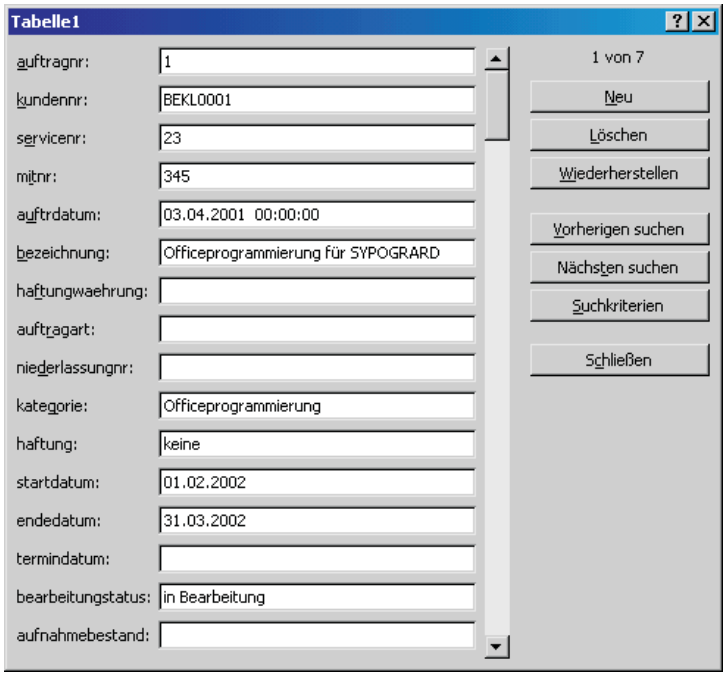


Abbildung 6.20: Excel als Eingabemaske

Im Beispiel aus Abbildung 6.20 wurde eine Abfrage auf die Tabelle *auftrag* der Beispieldatenbank ausgeführt. Mithilfe der Maske können Sie nun die Daten einsehen und modifizieren.

Externe Daten lassen sich auch mittels kleiner Programme (Makros), die in Visual Basic für Applikation (VBA) geschrieben wurden, in MS Excel einfügen. Auf diese Weise können Sie Excel-Tabellen als Frontend für Ihre Datenbank verwenden. Die Visual-Basic-Technologie, die dies ermöglicht, wird im Kapitel »Verbinden mit Datenquellen« beschrieben.

6.5.2 MS Access als Entwicklungswerkzeug

Auch wenn bei MS Access einiges an Funktionalität fehlt, die es zu einem richtigen DBMS machen würde, so kann es doch als hilfreiches Werkzeug bei der Entwicklung und Administration Ihres Datenbanksystems dienen. Nutzen Sie die einfache und überschaubare Funktionalität von MS Access beim Entwurf Ihrer Tabellen und Abfragen. Greifen Sie über eine Access-Instanz auf Ihre Datenquelle zu, indem Sie die Tabellen der Datenquelle mit MS Access verknüpfen. Die notwendigen Maßnahmen seien hier kurz zusammengefasst:

1. Starten Sie MS Access und legen Sie dort eine neue Datenbank an.
2. Wählen Sie Tabellen als Datenbankobjekt.
3. Klicken Sie auf NEU und im Kontextmenü auf TABELLE VERKNÜPFEN.
4. Im Dialogfeld VERKNÜPFEN geben Sie als Dateityp ODBC-Datenbanken an. (Wie Sie eine ODBC-Datenbank einrichten, können Sie in Kapitel 7 nachlesen.)
5. Wählen Sie die Datenquelle aus der Auswahlliste (siehe Abbildung 6.21).

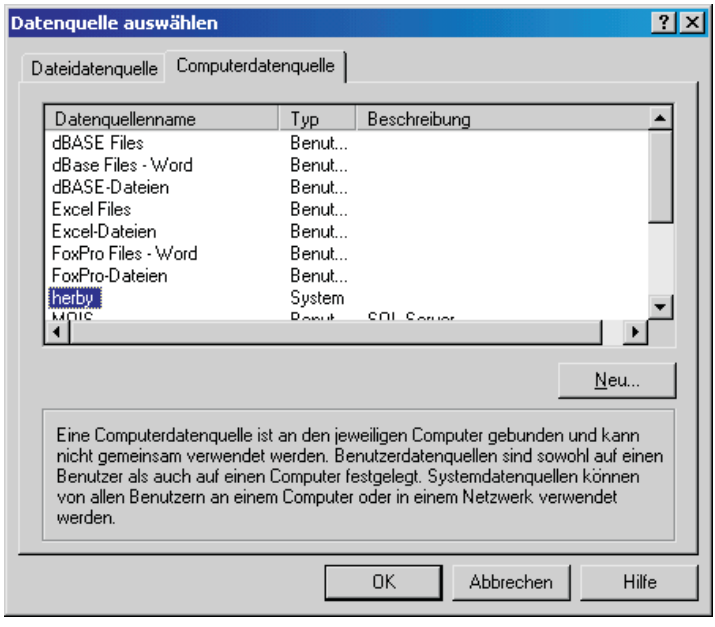


Abbildung 6.21: Dialogfeld für die ODBC-Verbindung

6. Treffen Sie eine Auswahl bzgl. der zu importierenden Tabellen.
7. Die Tabellen sind nun mit der aktuellen Access-MDB verknüpft. Sie erkennen den Unterschied zwischen verknüpften und nicht verknüpften Tabellen an dem Symbol vor dem Tabellennamen. (siehe Abbildung 6.22)

Mit den verknüpften Tabellen können Sie ebenso operieren wie mit Tabellen, die in Access angelegt wurden. Sie können diese Tabellen für Abfragen, Berichte oder Änderungen in der Tabellenansicht verwenden. Im Gegensatz zu den nicht verknüpften Tabellen, haben Sie jedoch keine Berechtigung, die Struktur der Tabellen zu ändern. Dies ist nur im Quellsystem möglich.

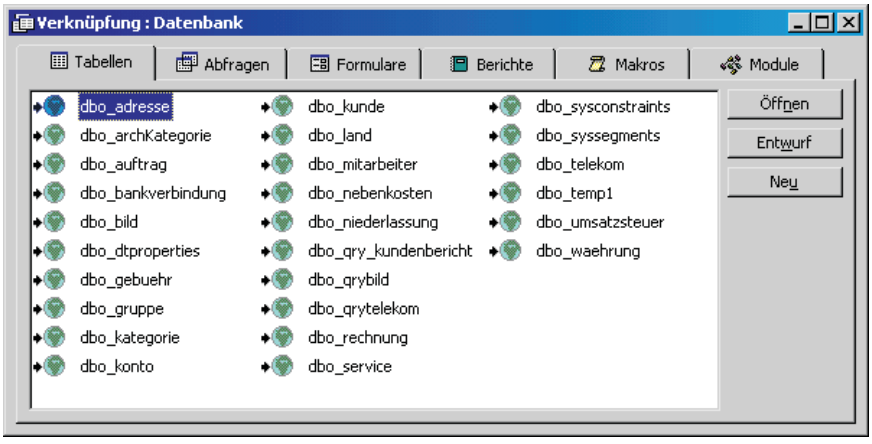


Abbildung 6.22: Verknüpfte Tabellen in Access

Öffnen Sie eine von den verknüpften Tabellen und ändern dort Daten in den verschiedenen Spalten, so wirkt sich die Änderung auch auf die Datenquelle aus. Ohne weiteren Programmieraufwand haben Sie sich so eine Benutzerschnittstelle geschaffen, mit der Sie Daten lesen und verändern können.

Die Tabellenstruktur mag nun für so manchen Anwender ungeeignet sein, um Daten einzugeben. Deshalb soll außerdem gezeigt werden, wie Sie mit wenig Aufwand eine Eingabemaske in Access erstellen können.

1. Erstellen Sie ein neues Formular, indem Sie beim Datenbankobjekt Formular auf NEU klicken.
2. Wählen Sie im geöffneten Dialogfeld den Formualar-Assistenten und die Tabelle *adresse* als Datenquelle (siehe Abbildung 6.23).
3. Wählen Sie im nächsten Schritt alle verfügbaren Felder, die im Formular enthalten sein sollen.
4. Geben Sie für das Layout EINSPALTIG an.
5. Übernehmen Sie die Standardeinstellung für den Stil.
6. Vergeben Sie einen Namen für das Eingabeformular.

Das waren schon alle notwendigen Schritte für die Erstellung einer simplen Eingabemaske in MS Access. Wenn Sie die Bezeichnungsfelder jetzt noch in der Entwurfsansicht ein wenig vergrößern, damit es nicht zu Abschneidungen kommt, so haben Sie bereits Ihre erste Eingabemaske erstellt (siehe Abbildung 6.24).

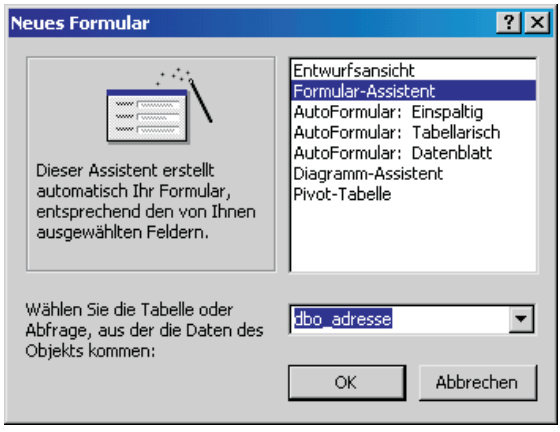


Abbildung 6.23: Dialogfeld NEUES FORMULAR

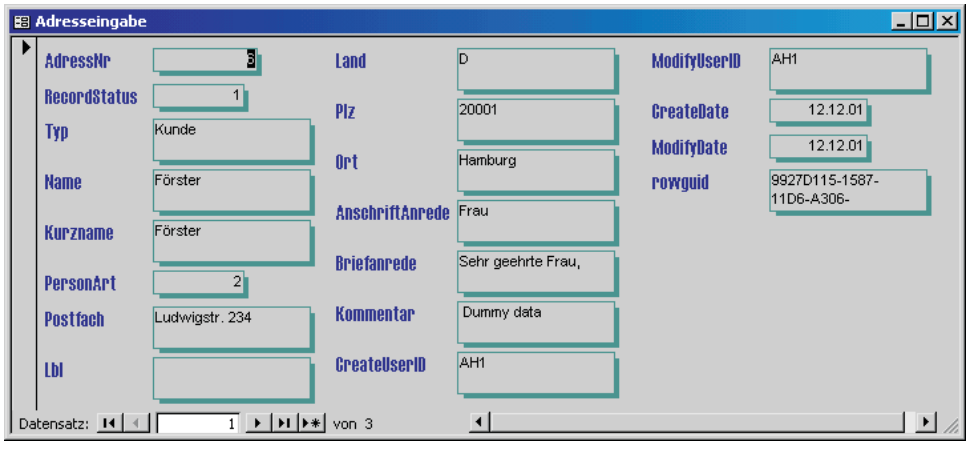


Abbildung 6.24: Eingabeformular in MS Access

Weitere Änderungen können nachträglich im Entwurfsmodus vorgenommen werden. Etwas komplizierter wird es, wenn sich ein Formular nicht ausschließlich auf eine Tabelle bezieht. Dann müssen Sie entweder mit Unterformularen arbeiten, oder Sie definieren zuvor eine Abfrage, welche die betroffenen Tabellen miteinander verknüpft.

6.5.3 PgAccess als Frontend für PostgreSQL

PgAccess ist wohl das am häufigsten verwendete graphische Tool für PostgreSQL, das seit neuestem mehrsprachig ist. Das Programm wurde in der Scriptsprache Tcl/Tk entwickelt. Mit PgAccess können Sie z.B. Tabellen und Sichten anlegen und verändern. Viele Arbeiten, die ansonsten etwas umständlich erscheinen, lassen sich in PgAccess

menügesteuert erledigen. Bevor Sie mit PgAccess arbeiten können, muss PostgreSQL, Tcl/Tk und selbstverständlich PgAccess auf Ihrem System installiert sein. Das Programm ist für die Plattformen Linux, FreeBSD, Solaris, HPUX und Windows 95,98,NT verfügbar. PgAccess verändert nicht die Art der PostgreSQL-Operationen, es erleichtert Ihnen jedoch die Arbeit damit. PgAccess bietet dem Anwender auch einige Funktionen für die Berichts- und Formularerstellung.

6.6 Einfache Masken selbst programmiert

Wollen Sie eine Datenbankapplikation neu erstellen, dann gehört das Design der Eingabemaske normalerweise zum Entwicklungsprozess. Welche Entwicklungsumgebung Sie für die Erstellung der Masken verwenden liegt an Ihren persönlichen Vorlieben. Die Entwicklerwerkzeuge von Microsoft haben neben der leicht zu bedienenden grafischen Oberfläche den Vorteil, dass Sie viele Technologien beinhalten, die auf Betriebssysteme, Officeanwendungen, Datenbanken und Browser aus dem Hause Microsoft zugeschnitten sind. In den nächsten Abschnitten wird gezeigt, wie Sie Eingabemasken in *Visual J++* und *Visual Basic* programmieren können. Eine andere Lösung wäre eine Dateneingabe über ein dynamisches HTML-Dokument in einem Browser. Mehr zu dieser Lösung und ihrer Umsetzung finden Sie im Kapitel »E-Commerce und Web«.

6.6.1 Benutzeroberfläche in Java

Dieser Abschnitt gibt Ihnen eine Einführung in die Gestaltung der Benutzeroberfläche mit der Entwicklungsumgebung *Visual J++* . Ein Dialogfeld für Erfassung von Mitarbeitern soll hier exemplarisch für das GUI der Beispieldatenbank erstellt werden. Die wesentlichen Schritte bis zur Benutzeroberfläche stellen sich wie folgt dar:

- Projekt anlegen,
- Dialogfeld einfügen,
- Steuerelemente aus der Werkzeugkiste einfügen,
- erzeugte Klassen einsetzen.

Projekt anlegen

Zu einem Java-Projekt gehören alle Elemente, die eine Anwendung ausmachen. So starten Sie ein neues Projekt in VJ++:

1. Starten Sie VJ++.
2. Wählen Sie in der Menüleiste FILE/NEW.

3. Klicken Sie im Kontextdialog auf PROJECT WORKSPACE.
4. Geben Sie im Dialogfeld NEW PROJECT WORKSPACE den Namen und das Verzeichnis für das neue Projekt an (siehe Abbildung 6.25).

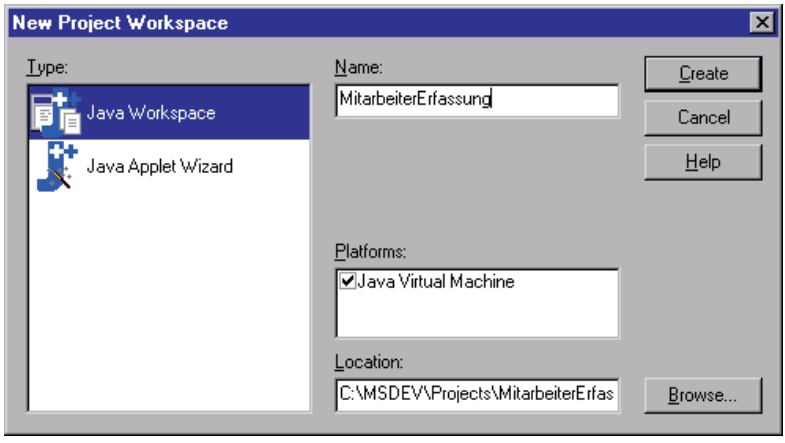


Abbildung 6.25: Der Resource Editor von VJ++

Dialogfeld einfügen

Mit dem *Resource Editor* von VJ++ können Benutzeroberflächen gestaltet werden. Sie können im Resource Editor sehr leicht Steuerelemente per Drag-and-Drop einfügen und die Eigenschaften festlegen. Die Steuerelemente werden in einer *Ressourcenvorlage* erzeugt und gespeichert.

1. Wählen Sie in der Menüleiste INSERT/RESOURCE...
2. Klicken Sie im Kontextdialog auf DIALOG (siehe Abbildung 6.26).
3. Klicken Sie auf die Befehlsschaltfläche OK.

Ein neues Dialogfeld wird nun eingefügt und es erscheint die Werzeugkiste, welche die verfügbaren Steuerelemente anzeigt.

Steuerelemente aus der Werkzeugkiste einfügen

Auf das zuvor eingefügte Dialogfeld können Sie die benötigten Steuerelemente platzieren. Fügen Sie vier Textfelder, sechs Bezeichnungsfelder, eine Befehlsschaltfläche und zwei Kombinationslisten ein. Die beiden Befehlsschaltflächen OK und ABBRUCH sind ohnehin schon im Dialogfeld enthalten. Über die Layout-Funktionen können Sie die Steuerelemente ausrichten. Ändern Sie die Beschriftung der Bezeichnungsfelder, indem Sie im Eigenschaftsfenster der Steuerelemente den Wert überschreiben, wie dies auch in Abbildung 6.27 gezeigt ist.

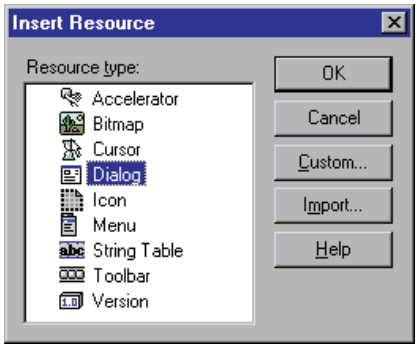


Abbildung 6.26: Einfügen eines Dialogfelds in VJ++

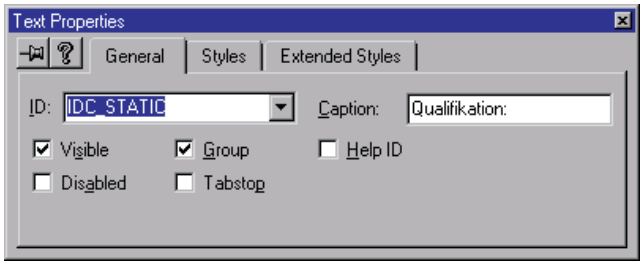


Abbildung 6.27: Eigenschaftenfenster in VJ++

Gestalten Sie das Layout auf dem Dialogfeld entsprechend der Darstellung in Bild 6.28.

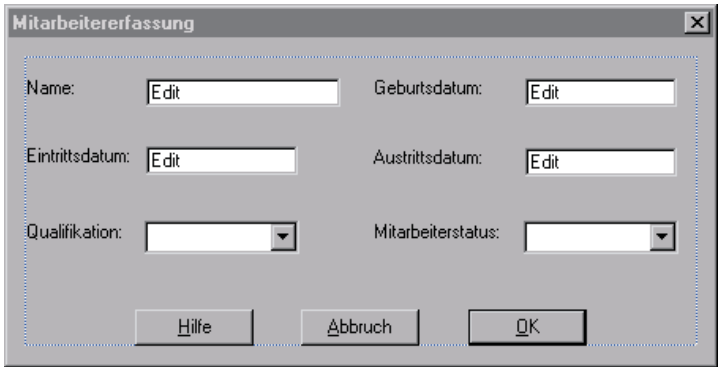


Abbildung 6.28: Dialogfeld mit Steuerelementen

Erzeugte Klassen einsetzen

Den bisher erzeugten Ressourcen fehlen noch die entsprechenden Java-Klassen. Um die fertiggestellten Ressourcen in Java-Code zu konvertieren, verwenden Sie den Java-Resource-Assistenten:

1. Starten Sie den *Java-Resource-Assistenten* über die Menüleiste: TOOLS/JAVA RESOURCE WIZARD...
2. Geben Sie im Dialogfeld einen Dateinamen für die Ressourcenvorlage an (siehe Abbildung 6.29).
3. Beim nächsten Schritt können Sie den vorgegebenen Namen für die Klassendatei übernehmen oder selbst einen Namen vorgeben.
4. Klicken Sie auf FERTIGSTELLEN.

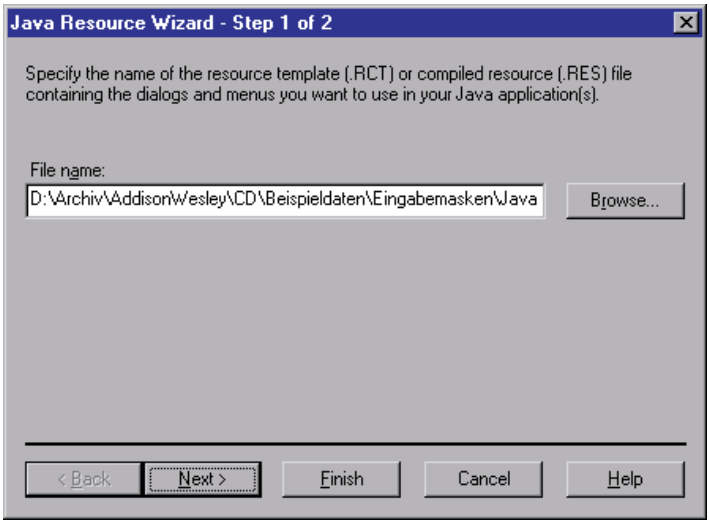


Abbildung 6.29: Der Java-Resource-Assistent

Es wurde eine Datei mit der Namenserweiterung *.JAVA erstellt. Sie können diese Datei öffnen und sich den erzeugten Code ansehen. Mit den Methoden der Steuerelemente können Sie die Eingaben der Benutzer auswerten. Im nächsten Kapitel werden Sie die Technologien kennen lernen, mit denen Sie Ihre Benutzeroberfläche mit der Datenbank verbinden.

6.6.2 Formulare mit Visual Basic

In Visual Basic werden die Eingabemasken oder Fenster auch *Formulare* genannt. Diesen Ausdruck kennen wir bereits von MS Access. Die Formulare bilden zusammen mit den Steuerelementen die Grundausstattung für das GUI-Design. Die Steuerelemente wenn auf den Formularen positioniert und anschließend über ihre Eigenschaften angepasst. Formulare sind Objekte mit *Eigenschaften*, *Methoden* und *Ereignissen*. Eigenschaften legen das Aussehen fest, Methoden bestimmen das Verhalten, und Ereignisse entscheiden über die Interaktion mit dem Benutzer. Sie können ein Formular an die Anforderungen Ihrer Anwendung anpassen, indem Sie die Eigenschaften des Formulars festlegen und einen Visual Basic-Code erstellen, der durch die Ereignisse des Formulars aktiviert wird. Die Verknüpfung zur Datenbank erfolgt entweder über eine direkte Verknüpfung der Steuerelemente mit der Datenquelle oder mittels ereignisgesteuerter Routinen. Im letzten Abschnitt wurde bereits gezeigt, wie einfache Ein- und Ausgabeelemente auf der Maske positioniert werden. In diesem Abschnitt soll eine weitere Möglichkeit vorgestellt werden. Die Ergebnismenge einer Abfrage soll in Tabellenform auf der Maske erscheinen. Wir benutzen dafür ein Steuerelement, das bisher nicht vorgestellt wurde, das *DataGrid-Steuerelement*.

DataGrid-Steuerelement

Mit dem Steuerelement *DataGrid*, können Datensätze in Reihen und Zeilen ausgegeben und bearbeitet werden. Jedem Feld wird dabei eine Zelle im DataGrid zugewiesen. Jede Zelle eines DataGrid-Steuerelements kann Textwerte, nicht aber verknüpfte oder eingebettete Objekte, enthalten. Sie können die aktuelle Zelle in Code festlegen, oder der Benutzer kann sie zur Laufzeit mit der Maus oder den Pfeiltasten ändern. Zellen können interaktiv durch Eingabe oder programmgesteuert geändert werden. Es ist möglich, Zellen einzeln oder zeilenweise auszuwählen.

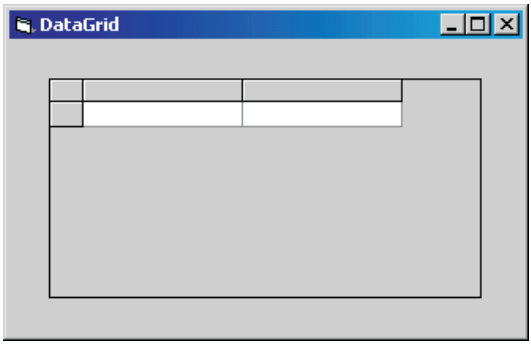


Abbildung 6.30: Das Steuerelement DataGrid

Wenn der Text einer Zelle zum Anzeigen zu lang ist, kommt es zu einem Zeilenumbruch und der Text wird in der nächsten Zeile derselben Zelle angezeigt. Damit der umbrochene Text angezeigt werden kann, müssen Sie Eigenschaften des DataGrid anpasssen. Zur Entwurfszeit können Sie die Spaltenbreite interaktiv ändern, indem Sie die Größe der Spalte ändern oder indem Sie die Spaltenbreite auf der Eigenschaftenseite ändern.

Telefonverzeichnis mit DataGrid

Das Steuerelement DataGrid wollen wir nun für ein elektronisches Telefonverzeichnis verwenden. Als Datenquelle soll uns dabei die Beispieldatenbank dienen. Über ein Textfeld soll der Suchbegriff eingegeben werden. Außerdem wird dem Benutzer über Kontrollkästchen die Möglichkeit eingeräumt, eine gezielte Suche entsprechend der gewählten Telekommunikations-Kategorie vorzunehmen. Wählt er z.B. die Kategorie *Fax*, so werden nur die Faxnummern in der Datenbank berücksichtigt. Die Eingaben des Benutzers (Textfeld und Kontrollkästchen) müssen ausgewertet werden und finden sich in der WHERE-Klausel der SQL-Abfrage wieder. Es sollen nun die notwendigen Schritte im Einzelnen beschrieben werden:

Erstellen Sie eine neues Projekt in Visual Basic und fügen Sie auf dem Formular zunächst die Steuerelemente ein, so dass das Formular wie in Abbildung 6.31 aussieht.

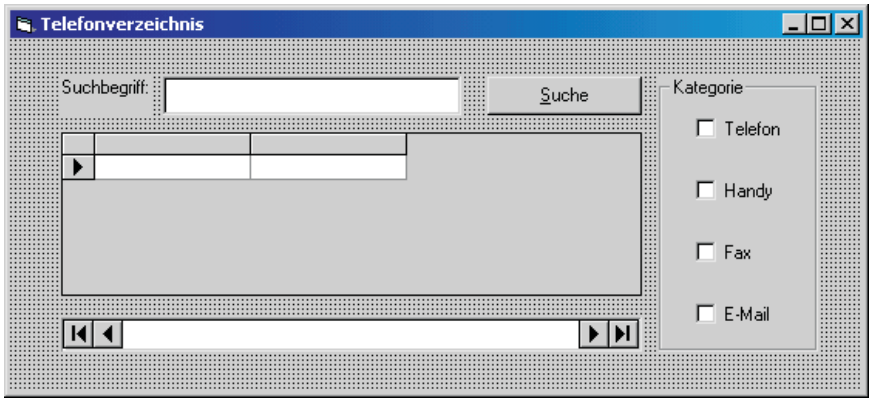


Abbildung 6.31: Benutzeroberfläche für das Telefonverzeichnis

Neben den bereits bekannten Steuerelementen *Bezeichnungsfeld*, *Textfeld*, *Befehlsschaltfläche*, *Rahmen*, *Kontrollkästchen* und *DataGrid* wurde in Abbildung 6.31 unter dem DataGrid ein weiteres Steuerelement eingefügt. Es handelt sich dabei um das *ADO-Datensteuerelement*. Damit Sie dieses Steuerelement einfügen können, müssen Sie es zunächst über das Dialogfeld KOMPONENETEN hinzufügen (siehe auch Abbildung 6.32).

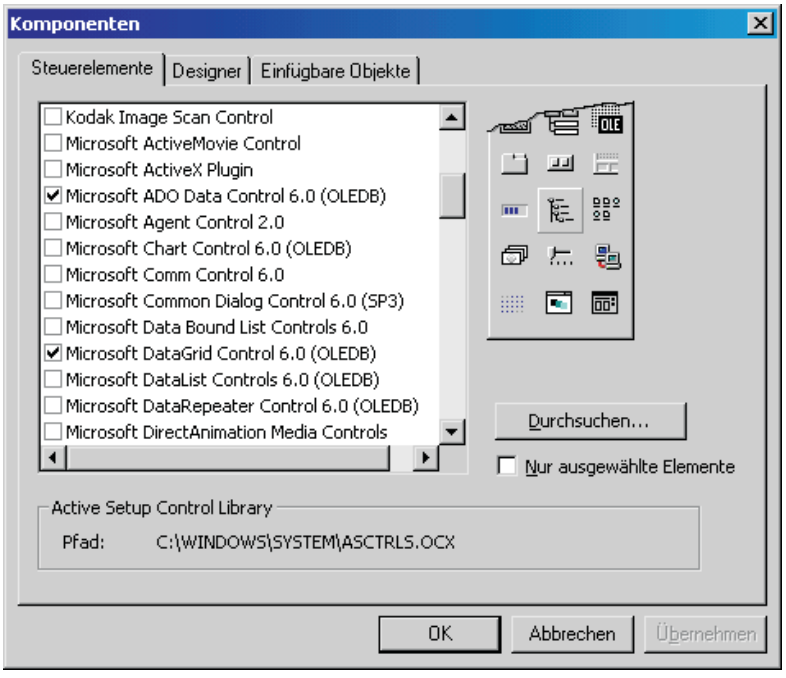


Abbildung 6.32: Komponenten hinzufügen in Visual Basic

Haben Sie alle Elemente auf dem Formular eingefügt, sollten Sie diese der Größe nach anpassen und über die Format-Funktionen ausrichten.

Legen Sie die Datenquelle fest, indem Sie dem ADO-Datensteuerelemment die Eigenschaft *ConnectionString* vergeben. Um dies zu tun, wählen Sie das Steuerelement aus und klicken Sie im Eigenschaftsfenster in das leere Feld rechts neben *ConnectionString*. (siehe Abbildung 6.33)

Im geöffneten Dialogfeld EIGENSCHAFTENSEITEN wählen Sie die zuvor definierte ODBC-Datenquelle aus. Wie Sie eine ODBC-Datenquelle definieren, erfahren Sie im nächsten Kapitel. Außerdem müssen Sie im ADO-Datensteuerelemment die *Record-Source-Eigenschaft* einstellen. Das ist in diesem Fall eine SQL-Anweisung, welche die Datensätze für das DataGrid erzeugt. Klicken Sie wieder in das leere Feld neben RecordSource und geben Sie unter BEFEHLSTEXT(SQL) im Dialogfeld EIGENSCHAFTENSEITEN die folgende Anweisung ein:

```
SELECT kurzname, kategorie, nummer FROM telekom ORDER BY kurzname
```

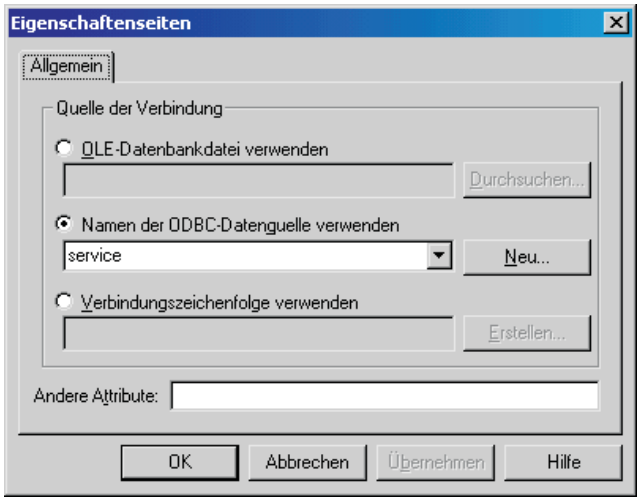


Abbildung 6.33: Verbindungsquelle in den Eigenschaftenseiten

Das Formular ist nun mit der Datenquelle verbunden. Im nächsten Schritt wird das DataGrid mit dem ADO-Datensteuerelement verknüpft. Wählen Sie das DataGrid im Entwurfsmodus aus und geben Sie für DataSource im Eigenschaftenfenster den Namen des ADO-Datensteuerelement an. Der Name sollte bereits in der Auswahlliste im DataSource-Feld verfügbar sein.

Wenn Sie die Anwendung jetzt ausführen (AUSFÜHREN/STARTEN), dann wird das DataGrid mit den Datensätzen aus der Tabelle *telekom* gefüllt. Die Suchfunktion und die Einschränkung über die Kategorie funktionieren allerdings noch nicht. Dafür ist ein wenig Programmcode erforderlich.

```
Dim rs AS Recordset
Dim strQuery As String
strQuery = "SELECT kurzname, kategorie, nummer FROM telekom"
IF txtSuchbegriff.Text <> "" THEN
strQuery = strQuery & " WHERE kurzname = '" & _ txtSuchbegriff.Text & "'"
END IF
SET rs = Adodc1.RECORDSET
Adodc1.RECORDSOURCE = strQuery
Adodc1.CAPTION = txtSuchbegriff.Text
Adodc1.REFRESH
```

In den ersten beiden Zeilen werden die Variablen deklariert. Der Variablen *strQuery* wird in der dritten Zeile im Programmcode eine SQL-Anweisung übergeben. Wird im Textfeld ein Suchbegriff vom Benutzer eingegeben, so wird die ursprüngliche SQL-Anweisung um eine WHERE-Klausel ergänzt. Dies geschieht innerhalb der IF-Anweisung. Die dynamisch erstellte Datensatzquelle wird dem ADO-Datensteuerelement

über die Eigenschaft RECORDSOURCE zugewiesen. Außerdem wird die Beschriftung des ADO-Steuerelementes mit dem Suchbegriff aktualisiert.

Sie können jetzt das Programm ausführen. Wenn Sie als Suchbegriff einen Namen eingeben, wird nach der Person in der Datenbank gesucht, so wie Sie das auch in der Abbildung 6.34 sehen können. In der tabellarischen Auflistung der Ergebnismenge können Sie direkt Daten verändern. Die Veränderung wird auch unmittelbar auf die Daten in der Datenbank übertragen.

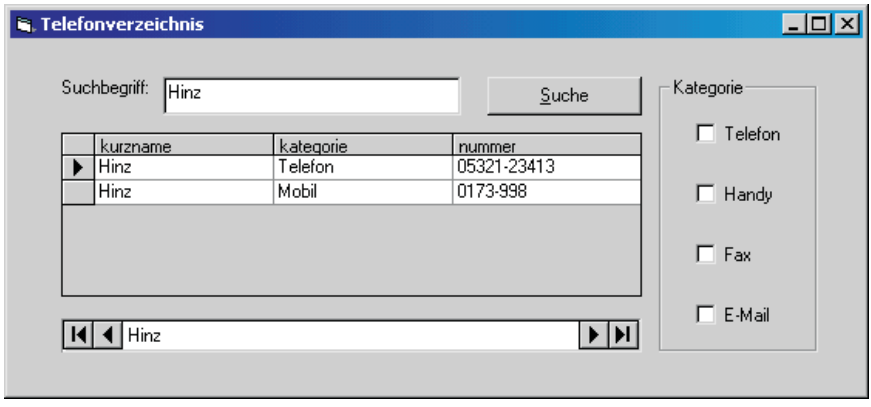


Abbildung 6.34: Das Telefonverzeichnis

Sie können den Programmcode erweitern, indem Sie nicht nur ganze Namen als Suchbegriff zulassen, sondern auch Namensmuster, wie z.B. »Her*«. Die Einschränkung über die Kategorie muss auch noch über eine SELECT-CASE-Anweisung ausgewertet werden.

7 Verbinden mit Datenquellen

Die Ausführungen im letzten Kapitel haben bereits angedeutet, wie wichtig die richtige Verbindung zwischen der Client-Anwendung und der Datenquelle ist. Damit die entwickelte Applikation nicht nur auf eine Datenquelle beschränkt ist, gibt es Standardtreiber, die den Zugriff auf alle namhaften Datenbanken ermöglichen. Diese müssen installiert und konfiguriert werden. In diesem Kapitel lernen Sie die verschiedenen Methoden kennen, mit denen Datenbanken angesteuert werden.

7.1 Die richtige Schnittstelle zwischen Programm und Datenquelle

Zwischen der Anwendung, welche die Daten präsentiert und der Datenbank, die als Datenquelle anzusehen ist, muss ein Austausch an Daten stattfinden. Bei RDBMS werden Anforderungen zwischen Client und Server als SQL-Anweisungen gesendet. Bei der Einführung von DBMS nutzten die Anwendungen in der Regel eingebettetes SQL, um auf eine Datenquelle zuzugreifen. Das eingebettete SQL ist zwar effizient, aber es erlaubt nicht, dynamisch auf Aktionen des Anwender zu reagieren. Das gesamte SQL wird zur Übersetzungszeit festgelegt und kann zur Laufzeit nicht geändert werden.

Die DBMS wurden weiterentwickelt und die Hersteller führten Schnittstellen ein, die es einer Anwendung ermöglichten, SQL-Anweisungen zur Laufzeit zu übersetzen. Man nennt diese Schnittstellen *APIs* (Application Programming Interfaces). Durch die Verwendung von *Programmbibliotheken* (DLLs) für jedes DBMS können Anwendungen geschrieben werden, die mit mehreren DBMS kommunizieren können, ohne dass sie neu übersetzt werden müssen. Ein großer Nachteil waren jedoch fehlende Standards. Jedes API unterstützt eigene Funktionen und Strukturen.

Über Datenbankschnittstellen und Netzwerkprotokolle ist es möglich, Anwendungen auf eine Vielzahl von Clientcomputern zu verteilen und verschiedene DBMS anzusteuern.

Eine Datenbankanwendung, die auf dem Client-Server-Modell basiert, besteht aus vier Funktionsgruppen. Auf der Seite des Client ist dies die Benutzerschnittstelle und die Anwendungslogik. Die DB-Schnittstelle bildet die verbindende Einheit zwischen Client und DBMS (siehe Abbildung 7.1).

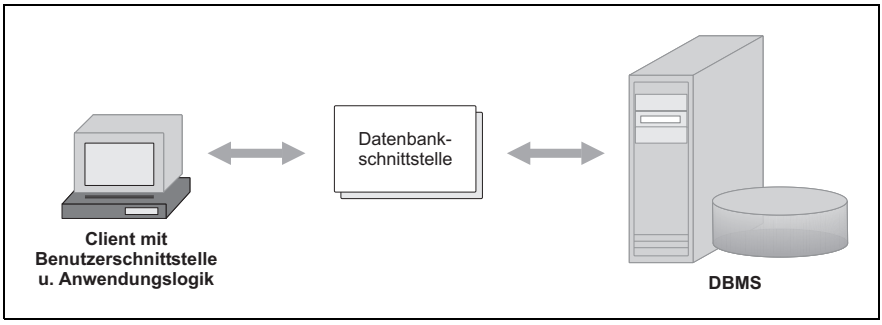


Abbildung 7.1: Client-Server-Modell mit Datenbankschnittstelle

Werden die vier Funktionsgruppen wie in Abbildung 7.1 auf Client und DB-Server verteilt, dann wird mit einer *2-Schichten-Architektur* gearbeitet. Um zu hohe Netzwerkbelastung zu vermeiden, wird auch oft eine *3-Schichten-Architektur* eingesetzt, bei der Anwendungslogik und DB-Schnittstelle auf einem extra Server ausgelagert werden.

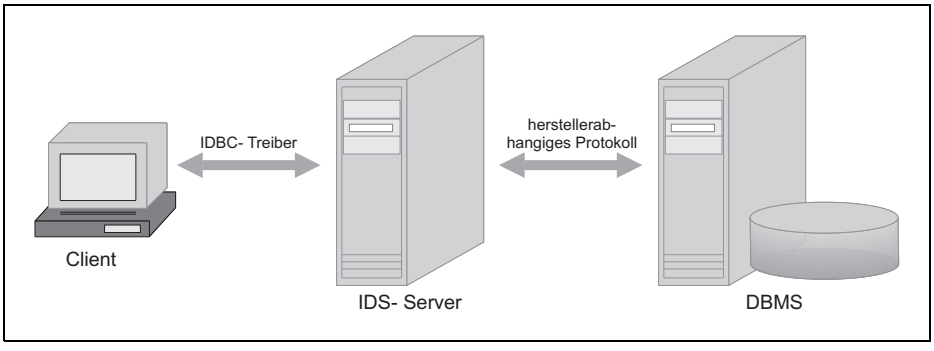


Abbildung 7.2: Schichten-Architektur

Insbesondere bei Web-Mechanismen (siehe Abbildung 7.2) kommt diese Architektur zum Einsatz. Unabhängig von der verwendeten Architektur ist eine DB-Schnittstelle notwendig. Die Wahl der Schnittstelle hängt im Wesentlichen von den anderen Komponenten, der Datenbankanwendung und den Betriebsystemen ab.

7.1.1 Open Database Connectivity (ODBC)

Die *Open Database Connectivity* (ODBC) ist eine Schnittstelle für den Zugriff auf Datenbanken. Unter Beteiligung von Microsoft wurde durch die SQL Access Group 1992 die ODBC-Schnittstelle definiert, welche in Analogie zu bereits bekannten Architekturen von Netzwerkzugängen und Hardwaretreibern in Betriebssystemen eine flexible Anbindung von Datenbanken an Neuentwicklungen oder bestehenden Anwendungsprogrammen zum Ziel hatte.

Durch die Anbieter der Datenbanken werden die entsprechenden ODBC-Treiber bereitgestellt. In den Betriebssystemen werden auch einige Standard-ODBC-Treiber mitgeliefert. Die ODBC-Schnittstelle kann als Baustein (SDK-Toolkit oder entsprechende API-Sammlung) bei den meisten Entwicklungssystemen in Anwendungen eingebunden werden.

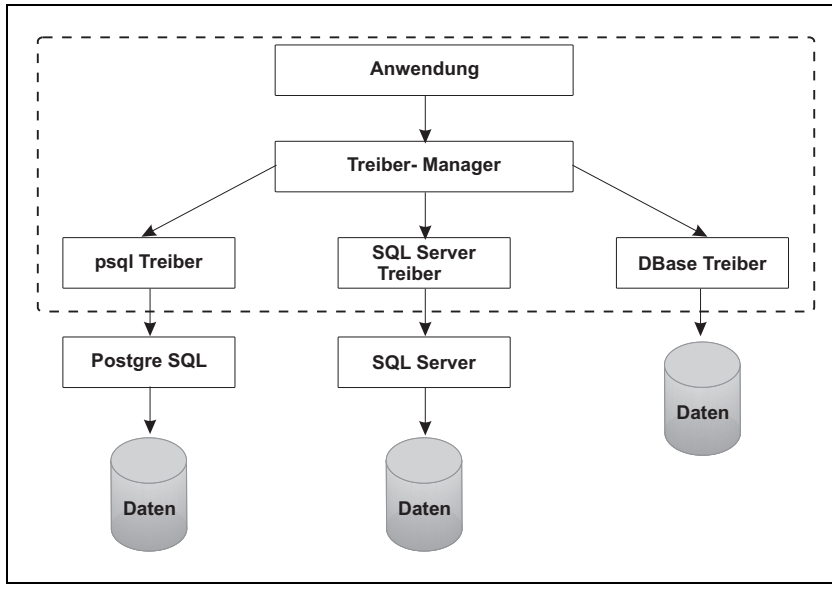


Abbildung 7.3: Architektur einer ODBC-Anwendung

Untersuchungen zur Performance von ODBC-Datenbanken zeigen, dass bei optimal konfigurierten ODBC-Treibern und Schnittstellen *kaum Geschwindigkeitsverluste* im Vergleich zum direkten Zugriff entstehen.

Durch die OBDC-Schnittstelle kann eine optimale Kombination von professionellen Datenbanken und leistungsfähiger Entwicklungssoftware für die Client-Anwendung realisiert werden.

Gleichzeitig kann auf verschiedene Datenquellen zugegriffen werden und auch kleinere Programme können durch die Einbindung von ODBC-Schnittstellen von ausgereiften Datenbanktechnologien Gebrauch machen.

Die rasante Entwicklung der Internetserver und Entwicklungswerkzeuge zeigt sehr deutlich, dass ODBC-Schnittstellen oder entsprechende API-Funktionen zur Standardausstattung gehören (siehe auch Kapitel 11).

Die Abbildung 7.3 zeigt die Architektur einer ODBC-Anwendung. Zur Architektur gehören vier Komponenten: die *Anwendung*, der *Treibermanager*, *ODBC-Treiber* und

Datenquelle. Von der Anwendung wird nach einer Benutzeraktion eine ODBC-Funktion aufgerufen, um SQL-Anweisungen abzusetzen. Der *ODBC-Manager* verwaltet die von der Anwendung geforderten Treiber und verarbeitet die ODBC-Funktionsaufrufe. Der ODBC-Treiber stellt die Funktionalität für die spezielle Datenquelle in Form einer DLL zur Verfügung. Ist es erforderlich, dass Daten in das herstellerspezifische Format umgeformt werden, so ist der Treiber auch dafür verantwortlich. DBMS wie MS SQL Server oder PostgreSQL bilden die Datenquelle.

7.1.2 Component Object Model (COM)

Das *Component Object Model* ist ein Ansammlung von *Spezifikationen*, *Datenstrukturen* und *Schnittstellen*. Das Zusammenspiel der verschiedenen Komponenten wird auf binärer Ebene geregelt. Mit diesem Modell soll plattformunabhängige Softwareentwicklung gewährleistet werden. COM ist die Basis für andere Technologien, wie *OLE DB* oder *ADO*. Mit COM wurde keine objektorientierte Sprache, sondern ein neuer Standard geschaffen. Es werden COM Objekte beschrieben, welche Daten beinhalten. Mit den definierten Schnittstellen kann über Methoden der Schnittstellen auf Objekte verschiedener Herkunft zugegriffen werden. Die einzige Anforderung die COM an die verwendete Programmiersprache stellt, ist dass diese eine *Zeigerstruktur* unterstützen, welche Funktionsaufrufe auslösen können. Die Liste der nutzbaren Programmiersprachen ist dann auch lang und reicht von C über Java und Smalltalk bis Basic.

Die Entwicklung von komponentenbasierter Software auf der Basis von COM soll zukünftig durch eine *COM+* genannte Erweiterung deutlich vereinfacht werden. Gegenwärtig erfordert beispielsweise jede Komponente, die Automation unterstützen soll, die manuelle Implementierung von Schnittstellen. Derartige wiederkehrende Aufgaben sollen zukünftig durch von COM+ bereitgestellte Dienste erledigt werden.

7.1.3 Object Linking and Embedding Database (OLE DB)

Mit *OLE DB* setzt Microsoft eine neue Treibergeneration ein, um nicht nur relationale DBMS ansteuern zu können, sondern auch objektorientierte. Zu den Datenquellen, die mit OLE DB verwaltet werden können, gehören neben den relationale Datenbanken, E-Mail-Dateien, unstrukturierte Dateien und Dateien mit Tabellendaten. Es handelt sich bei OLE DB um ein API, welches auf COM basiert. OLE verbindet neue Technologien und bietet dem Entwickler mehr Flexibilität beim Entwurf seiner Anwendung.

OLE DB definiert eine Sammlung von COM-Schnittstellen, die verschiedene Datenbankverwaltungssystemdienste umfassen. Diese Schnittstellen ermöglichen den Entwurf von Softwarekomponenten, die solche Dienste implementieren. OLE DB-Komponenten bestehen aus *Datenprovidern*, die Daten beinhalten und zur Verfügung stellen, Datenconsumern, die Daten verwenden, und Dienstkomponenten, die Daten verarbeiten und weiterleiten (z.B. Abfrageprozessoren und cursor engines). OLE DB-

Schnittstellen unterstützen die reibungslose Integration von Komponenten, so dass die Hersteller von OLE DB-Komponenten Produkte von hoher Qualität schnell auf den Markt bringen können. Darüber hinaus beinhaltet OLE DB eine Brücke zu ODBC, um eine kontinuierliche Unterstützung für die Vielzahl von ODBC-relationalen Datenbanktreibern zu ermöglichen, die zur Zeit verfügbar sind.

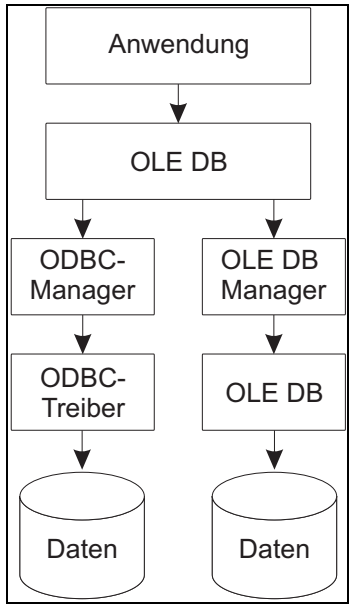


Abbildung 7.4: Architektur einer OLE DB-Anwendung

7.1.4 Java Database Connectivity JDBC

JDBC ist ein plattformunabhängiger Datenbanktreiber von der Firma Sun für Java. JDBC benötigt die direkte Ausführung von SQL-Anweisungen und wird deshalb auch *als Low-Level-API* bezeichnet. Klassen zur Ausführung von SQL-Anweisungen, für die Datenbankverbindung, zur Verwaltung der Treiber oder für den Zugriff auf Ergebnismengen gehören ebenso wie die Schnittstellen zum Gesamtpaket von JDBC.

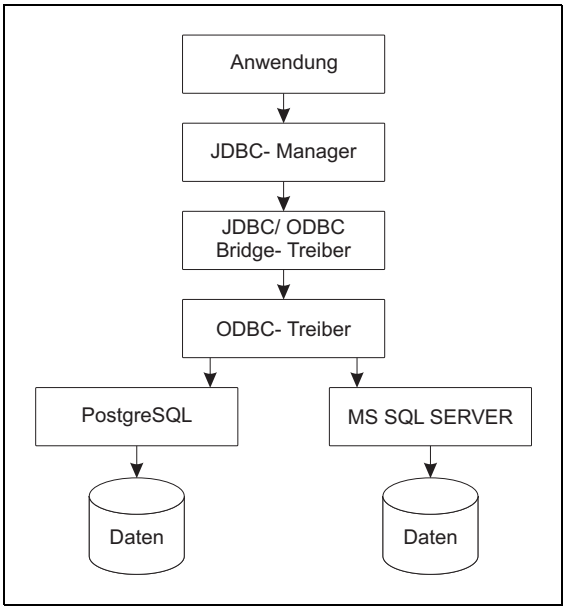


Abbildung 7.5: Verbindung zwischen Anwendung und Datenbank über JDBC

7.2 Installation der Treiberdateien

Voraussetzung für die Treiberinstallation ist das Vorhandensein eines Treibermanagers. Beim Treibermanger handelt es sich um eine DLL, die von Microsoft als Teil der ODBC-Installation bereitgestellt wird. Die eigentliche Aufgabe des Treibermanagers ist, die jeweils benötigten ODBC-Treiber zu laden oder zu entladen.

Wenn die Anwendung die Liste mit den Namen der installierten Datenquellen anfordert, ermittelt der Treibermanger diese Information aus der Registry und gibt eine Liste mit verfügbaren Datenquellen zurück.

Darüber hinaus verarbeitet der Treibermanager mehrere ODBC-Initialisierungsaufrufe sowie Eintrittspunkte in ODBC-Funktionen für die einzelnen Treiber und bietet eine Parameter- und Sequenzauswertung für ODBC-Funktionsaufrufe.

Der ODBC-Treiber realisiert den Großteil der Funktionalität für eine Datenquelle. Bei dem Treiber handelt es sich um eine DLL, die ODBC-Funktionsaufrufe implementiert und mit der Datenquelle interagiert oder die eigentliche Logik für die direkte Arbeit mit den Datendateien enthält. Wenn eine Anwendung eine Abfrage für die Datenquelle ansetzt, interpretiert der Treiber sie und sendet sie an die Datenquelle. Müssen die Daten dabei in ein oder aus einem anderen Format konvertiert werden, ist der Treiber dafür verantwortlich.

Die Installation der Treiber ist für den SQL Server keine allzu große Herausforderung und wird üblicherweise automatisch bei der Installation des DBMS durchgeführt. Deshalb wird die Installation des Treibers am Beispiel des psql-Treibers für PostgreSQL gezeigt.

Installation unter Linux:

1. Wechseln Sie in das Verzeichnis SRC/INTERFACES/ODBC und installieren Sie den Treiber, den Sie unter der Adresse, *http://www.iodbc.org*, herunterladen können.
2. Die Konfigurationsdatei odbcinst.ini wird in das Verzeichnis DIRECTORY /USR/LOCAL/PGSQL/ETC/ geschrieben.
3. Installieren Sie den ODBC-Erweiterungskatalog, durch Ausführung der folgenden Anweisung:

```
psql -d template1 -f LOCATION/odbc.sql
```

Installation unter Windows:

Entpacken Sie die Datei PSQLODBC-07_01_0009.ZIP, die Sie auf der Begleit-CD finden, in einem temporären Verzeichnis und starten Sie anschließend die Setup-Routine, durch einen Doppelklick auf die Datei PSQLODBC.EXE. Folgen Sie den Hinweisen des Installations-Assistenten (siehe Abbildung).

Abbildung 7.6: Installations-Assistent von PSQLODBC

7.3 Konfiguration oder Anmelden der Datenquelle bei der Datenbankschnittstelle

Sind die notwendigen Treiber für das DBMS installiert, dann kann die Datenbank bei der Schnittstelle angemeldet werden. Der Vorteil der Schnittstelle ist, dass durch sie die verschiedensten Datenquellen mit der Anwendung zusammenarbeiten können. Die Schnittstelle benötigt jedoch Informationen über die speziellen Datenquellen. Am Beispiel des zuvor installierten PsqlODBC-Treiber soll die Konfiguration mit dem ODBC-Datenquellen-Administrator beschrieben werden.

1. Starten Sie den ODBC-Manger START/EINSTELLUNGEN/SYSTEMSTEUERUNG/ ODBC-DATENQUELLEN.
2. Klicken Sie auf die Registerkarte SYSTEM-DSN.
3. Klicken Sie auf die Befehlsschaltfläche HINZUFÜGEN.
4. Suchen Sie im nächsten Dialogfeld den PostgreSQL-Treiber aus der Auswahlliste.

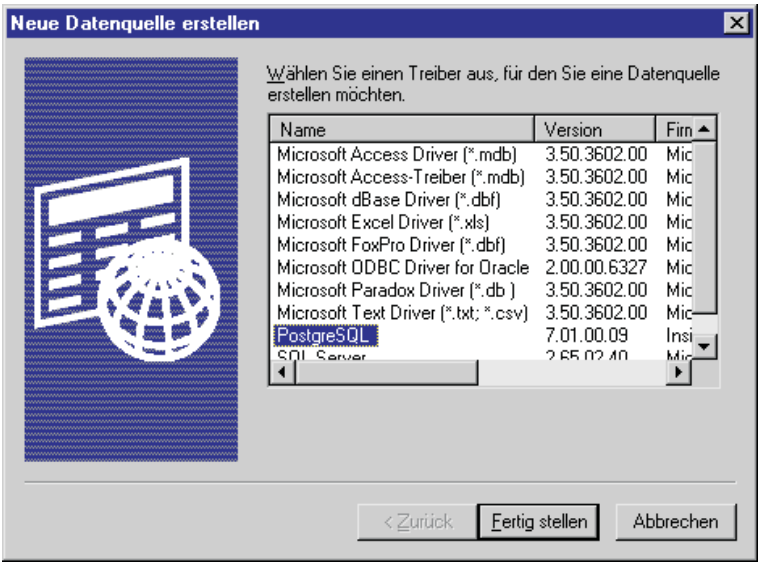


Abbildung 7.7: Auswahl des Treibers im ODBC-Manager

5. Geben Sie die Daten für Ihre Datenbank an (siehe Abbildung 7.8)
6. Klicken Sie auf FERTIGSTELLEN, um die Datenbank als Systemdatenquelle hinzuzufügen (siehe Abbildung 7.9).

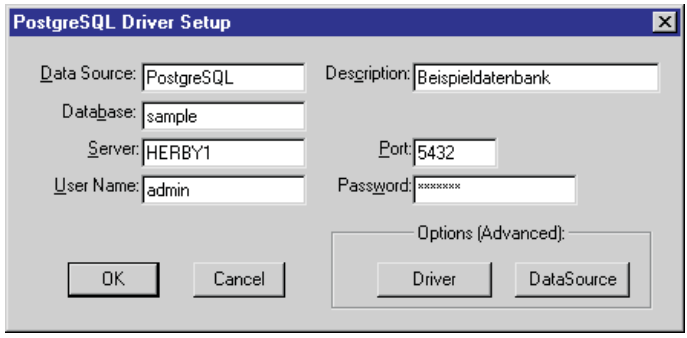


Abbildung 7.8: Konfiguration des Treibers im ODBC-Manager

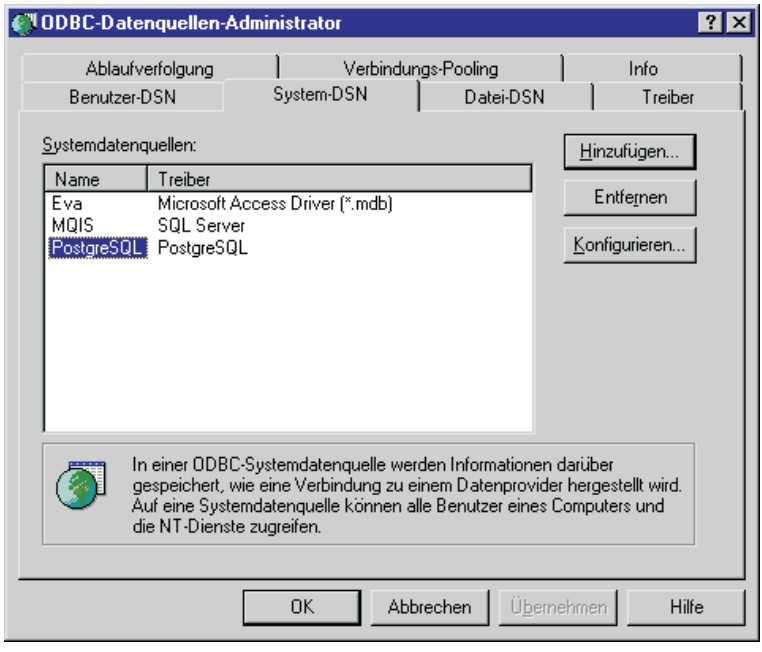


Abbildung 7.9: Eintrag für die Datenbank im ODBC-Manager

Ihre Datenbank kann nun über die Schnittstelle angesteuert werden. Sie können nun eine Verbindung von Ihrer Anwendung oder Standardanwendungen zur Datenbank mit ODBC erstellen. Die gemachten Einstellungen können nachträglich geändert werden. Öffnen Sie auch dafür den ODBC-Datenquellen-Administrator und klicken Sie auf die Befehlsschaltfläche KONFIGURIEREN.

7.4 Datenzugriffsobjekte der Programmiersprache

Bisher haben wir gelernt wie man eine Datenbank mit ihren Komponenten erstellt, wie man eine Eingabemaske erzeugt und wie eine Schnittstelle zwischen Datenbank und Eingabemaske aufgebaut wird. Das letzte Glied in der Kette der Datenbankanwendung sind die *Datenzugriffsobjekte* der Programmiersprache. Ihre Anwendung benötigt Datenbankfunktionen, mit denen die von der Schnittstelle gelieferten Werte innerhalb der Anwendung richtig verarbeitet werden können. So kann man die Datenzugriffsobjekte der Programmiersprache auch als Teil der Schnittstelle zwischen Programm und Datenbank sehen. In den folgenden Abschnitten sollen die Modelle zur Datenbankunterstützung in ihrer historischen Reihenfolge betrachtet werden.

7.4.1 Data Access Objects (DAO)

Data Access Objects (DAO) wurden zuerst mit Microsoft Access und Visual Basic eingeführt. DAO wurde entwickelt, um Datenbankfunktionen und -operationen innerhalb eines Objekts einzuschließen. DAO verschafft Ihnen Zugriff auf ODBC-Datenbanken.

DAO ist eine auf OLE basierende Anwendungsprogrammierschnittstelle (API). Im Allgemeinen bieten die DAO-Datenbankklassen umfangreichere Datenbankfunktionen als ODBC-Datenbankklassen, die erstmals in MFC (Microsoft Foundation Class) eingeführt wurden. DAO kann über ODBC auf andere Datenbanktypen zugreifen.

Nicht alle DAO-Objekte werden in MFC offengelegt, obwohl die meisten der DAO-Funktionen verfügbar sind.

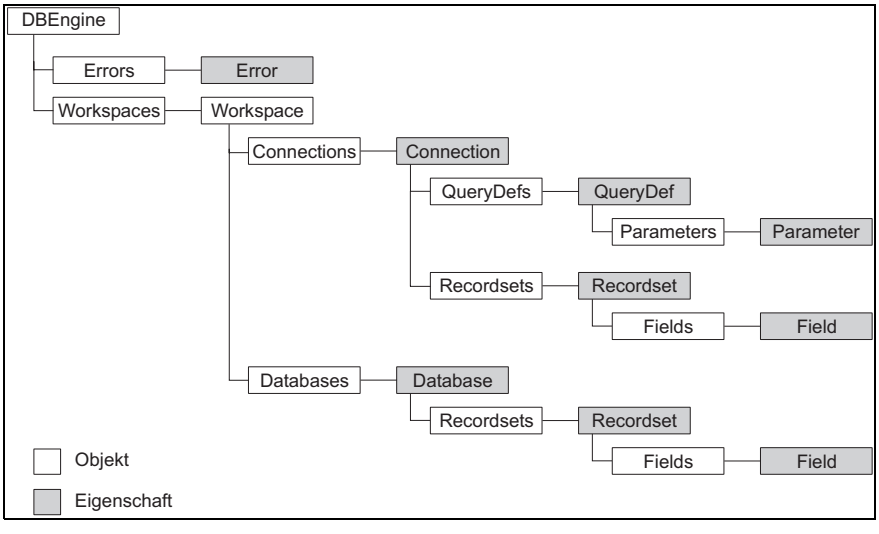


Abbildung 7.10: DAO-Modell

In Abbildung 7.10 sehen Sie die Objekthierarchie des DAO-Modells. An oberster Stelle steht das DBEngine-Objekt, welches für die *Jet-Engine* steht. Die Jet-Engine ist eine Ansammlung von DLLs, die alle Funktionen und Optionen bereitstellen, die der Entwickler beim DAO-Modell einsetzen kann. Es folgen weitere Objekte, wie das Objekt Workspace, welches die Plattform für Datenbanksitzungen darstellt. Die Beispieldatenbank (MS Access) könnte mit den Anweisungen, wie wir sie im Beispiellisting sehen, geöffnet werden.

Beispiel 7.1:

```
Dim db As Database
Set db = DBEngine.Workspaces(0).OpenDatabase("BeispielDb.mdb")
```

Sie können nun alle elementaren Datenbankoperationen mit DAO ausführen. Wenn Sie einige Datensätze aus der Datenbank abrufen, können Sie das Objekt *Recordset* verwenden. Recordsets sind elementar wichtige Objekte für den Datenbankprogrammierer. Dateninhalte aus der Datenbank werden, für die Verweildauer der betreffenden Datenbankoperation, in Recordsets geladen. Mögliche Datenänderungen durch den Anwender werden zunächst am Recordset ausgeführt und später ggf. in die Datenbank geschrieben.

Beispiel 7.2:

```
Dim rs As Recordset
Dim sSql As String
sSql = "SELECT * FROM adresse WHERE adressnr = 3"
Set rs = db.OpenRecordset(sSql, dbOpenDynaset)
```

In dem Beispiel 7.2, wird zur Bildung eines Recordset eine SQL-Anweisung verwendet. Im Recordset stehen nach erfolgreicher Ausführung der Prozedur alle Datensätze aus der Tabelle *adresse*. Voraussetzung für das Erstellen eines Recordset ist die Existenz eines gültigen Database-Objekts.

```
txtName = rs.Fields("name").Value
```

In der angegebenen Programmzeile wird der Wert des Feldes *name* vom Recordset an das Steuerelement *txtName*. Dieses Steuerelement sei ein Textfeld auf der Eingabemaske, das nach Ausführung den Wert für das Feld *name* aus der Datenbank anzeigt.

7.4.2 Remote Data Objects (RDO)

Remote Data Objects (RDO) sind die Nachfolger von DAO. Diese Objekte stellen eine bessere Lösung für den Datenbankzugriff via ODBC dar und erweitern darüber hinaus die Reichweite dieser Objekte bis zum Server. Die JET-Engine wird nicht verwendet. Durch einen stark an ODBC angelehnten Aufbau wird nahezu die Geschwindigkeit von ODBC erreicht. RDO-Objekte können mit ODBC-Funktionsaufrufen kombiniert werden.

RDO ist eine dünne Objektmodellschicht oberhalb der ODBC-API. Ein Großteil der Funktionalität von RDO hängt vom ODBC-Treiber und dem Datenbankmodul ab. Der Datenzugriff über RDO ist nur für relationale ODBC-Datenbanken gedacht.

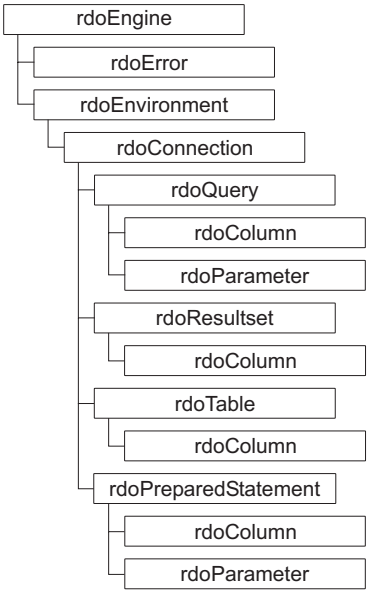


Abbildung 7.11: Das RDO-Modell

Die zahlreichen Objekte des Modells sind in Abbildung 7.11 zu sehen. Das rdoEngine-Objekt stellt dabei die Basis dar. Greift eine Anwendung das erste Mal auf RDO zu, so wird das Objekt *rdoEngine* automatisch erstellt. Auf weitere Objekte mit ihren Methoden und Eigenschaften kann der Entwickler zurückgreifen, um auf Daten des DBMS zuzugreifen und Fehler abzufangen.

Wollen Sie beispielsweise mittels RDO eine Verbindung zur Beispieldatenbank auf dem SQL Server aufbauen, dann würde der Programmcode wie folgt aussehen:

Beispiel 7.3:

```
Sub Verbindungsaufbau()
Dim rdoVerbdg As New rdoConnection
Dim sStr As String

sStr =   "UID=;PWD=;Database=Sample;" _
       & "Server=SEQUEL;Driver={SQL Server}" _
       & "DSN='';"
With rdoVerbdg
     .Connect = sStr
     .CursorDriver = rdUseODBC
     .LoginTimeout = 5
     .EstablishConnection rdDriverNoPrompt, True
End With

End Sub
```

Mit der angegebenen Routine wird das Objekt *rdoConnection* angelegt und Sie können im weiteren Verlauf der Anwendung über dieses Objekt Tabellen oder Sichten der Datenbank abfragen bzw. verändern.

7.4.3 ActiveX-Datenobjekte (ADO)

Die *ActiveX-Datenobjekte (ADO)* bilden eine abstrakte Datenschnittstelle von Microsoft, die den Entwickler vollständig von den zugrundeliegenden Technologien OLE DB und ODBC abschirmt. ADO bietet ein offenes Datenzugriffsmodell auf Anwendungsebene, welches den Entwicklern das Erstellen von Datenbankanwendungen für OLE DB-Daten in einer beliebigen Programmiersprache ermöglicht. Durch ADO steht dem Entwickler ein Zugriff auf eine größere Anzahl von Datentypen als je zuvor zur Verfügung. Für die Erstellung komplexer Anwendungen wird wesentlich weniger Zeit benötigt.

Mit ADO können Applikationen über jeden OLE DB-Provider auf Daten eines Datenbankservers zugreifen und diese verändern. (siehe Abbildung 7.12)

ADO arbeitet mit jeder ODBC-kompatiblen Datenbank. ADO ist zwar der Nachfolger von RDO, aber die beiden Modelle sind nicht identisch. ADO erweitert die Funktionalität von RDO auf das Internet. Sie unterscheiden sich darin, dass Sie bei ADO keine Objekthierarchie erstellen müssen, um bestimmte Kommandos auszuführen. Alle Objekte innerhalb des ADO-Modells können als einzelne Instanz erzeugt werden.

Mit ADO ist es möglich, die unterschiedlichsten Arten von Datenquellen anzusprechen. Die größte Rolle spielen dabei relationale Datenbanken, aber auch auf nicht relationale Daten kann mit ADO zugegriffen werden.

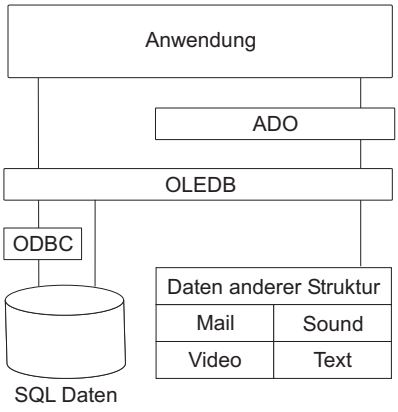


Abbildung 7.12: Architektur für eine ADO-Anwendung

ADO gestattet Ihnen, ActiveX-Scripts zur Herstellung der Verbindung zu Ihrer Datenbank zu verwenden. Sie können diese Scriptsprache auch zum Festlegen der Eigenschaften und Methoden eines ADOs benutzen. Das Objekt unterstützt verschiedenste Datentypen, wie Bilder und große Binärobjekte (BLOBs). ADOs ermöglichen außerdem das Arbeiten mit *Tansaktionen*, *verschiedenen Cursorn*, *Fehlerbehandlungsroutinen* und die Verwendung von *gespeicherten Prozeduren*.

Vorteile von ADO	**Nachteile von ADO**
Geringer Bedarf an Speicherressourcen, einfache Bedienung, bei relativ hoher Geschwindigkeit.	Je nach Datenquelle unterschiedliche Datentypen, Cursor belasten durch Nachladen der Daten das Netz und den Datenbankserver.

Tabelle 7.1: Vor- und Nachteile der ActiveX-Datenobjekte

Die Abbildung 7.13 stellt das Objektmodell für ADO vor.

Sicherlich haben Sie festgestellt, dass es im Vergleich zu DAO und RDO mit weniger Objekten auskommt. Die Säulen dieses Modells sind die Objekte *Command* für die Abfragebeschreibung, *Connection* mit den Verbindungsinformationen, und *Recordset* mit der Ergebnismenge der Abfragen.

Beispiel 7.4:

```
Set cn = New ADODB.Connection
With cn
    .Provider = "Microsoft.Jet.OLEDB.3.51"
    .ConnectionString = "Data Source = 'BeispielDb.mdb'"
    .Open
End With
```

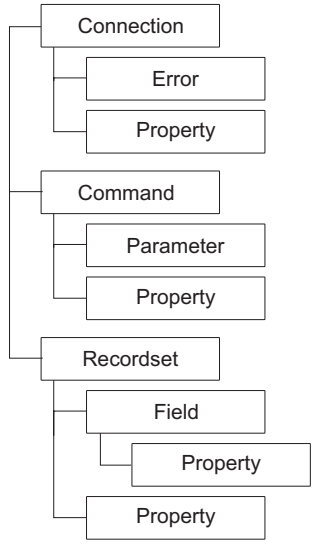


Abbildung 7.13: Das Objektmodell von ADO

Das Programmlisting aus Beispiel 7.4 baut über das Connection-Objekt eine Verbindung zur Beispieldatenbank auf. Mit Angaben zum Provider und Namen der Datenquelle wird die Verbindung aufgebaut. Wie schon beim RDO-Modell beschrieben, kann nun ein Recordset-Objekt geöffnet und mit Daten aus der Datenbank gefüllt werden. Die Verbindung sollte mit einer CLOSE-Anweisung geschlossen werden, wenn Sie nicht mehr benötigt wird.

8 Datenbank verwalten

Nach dem Erstellen der Datenbank beginnt der administrative Teil der Arbeit. Die Performance der Datenbank muss getestet werden, Funktionalität im Multiuser-Modus muss überprüft werden, Fremddaten müssen in die Datenbank übernommen werden, Daten sollen auf mehrere Standorte verteilt werden. Diese Szenarien werden in diesem Kapitel beschrieben und Lösungsvorschläge anhand der Beispieldatenbank gemacht.

8.1 Optimieren der Indizes

Damit eine optimale Leistung dauerhaft gewährleistet ist, müssen erstellte Indizes korrekt verwaltet werden. Indizes können ohne Auswirkungen auf das Datenbankschema oder den Anwendungsentwurf gelöscht, hinzugefügt und geändert werden. In der Entwurfsphase sollten Sie mit verschiedenen Indizes experimentieren. Viele DBMS bieten Werkzeuge, mit denen die Indizes optimiert und Abfragen analysiert werden können.

8.1.1 Analyse der eingesetzten Indizes

Dieser Abschnitt soll zeigen, wie es mit den Werkzeugen eines DBMS möglich ist, Leistungen und insbesondere *Abfrageleistungen* zu analysieren. Der SQL Server hat zwei Module, die einerseits die Überwachung von Datenbankereignissen und andrerseits eine Analyse ermöglichen. Deshalb wird auch für diese Aufgabe ein Anwendungsbeispiel auf dem SQL Server durchgeführt. Aufgrund der Aufzeichnungen, die bei der Überwachung durch den *SQL Server Profiler* erstellt wurden, kann eine Analyse der SQL-Abfragen und der Effektivität der Indizes mit dafür vorgesehenen Assistenten erfolgen.

Ablaufverfolgung mit dem SQL Server Profiler

Mit dem *SQL Server Profiler* ist es möglich, die verschiedenen Aktivitäten an der Datenbank zu überwachen. Dadurch eignet sich der Profiler sowohl zur Leistungsoptimierung, als auch zur Fehlersuche. Es sollen die Aktivitäten an der Beispieldatenbank *sample* überwacht und aufgezeichnet werden, damit die Aufzeichnungen später analysiert werden können.

1. Starten des SQL Server Profiler über START/PROGRAMME/MICROSOFT SQL SERVER/PROFILER.
2. Im Menü DATEI wählen Sie NEU/ABLAUFVERFOLGUNG.
3. Erstellen Sie die Verbindung zum Datenbankserver.
4. Im Dialogfeld ABLAUFVERFOLGUNGSEIGENSCHAFTEN (Trace properties) geben Sie in der Registerkarte ALLGEMEIN den Namen für die Ablaufverfolgung an (siehe Abbildung 8.1).
5. Machen Sie eine Angabe zu dem Verzeichnis, in dem die Ablaufverfolgung gespeichert werden soll.

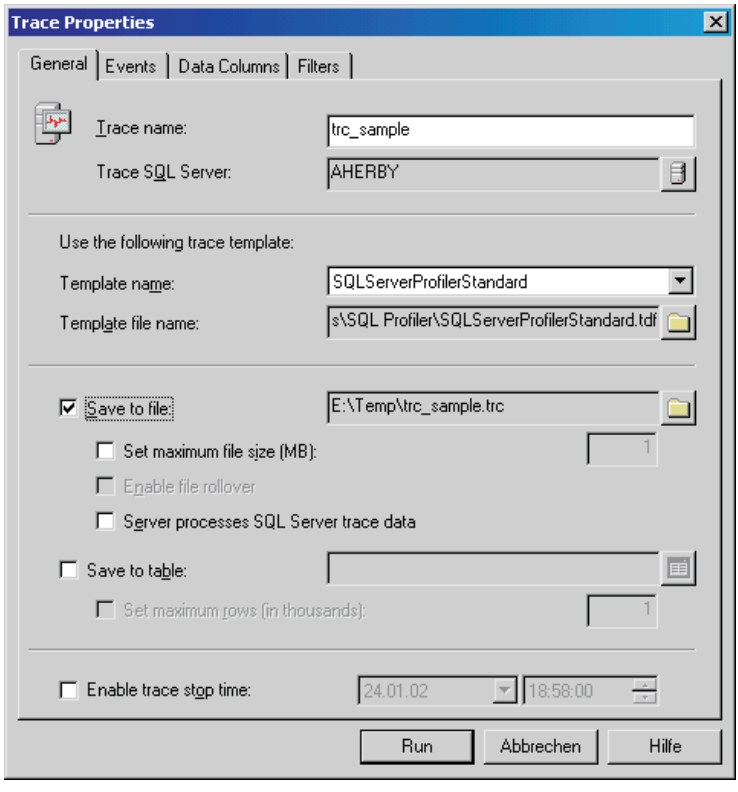


Abbildung 8.1: Angabe der Ablaufverfolgungseigenschaften

6. In der Registerkarte EREIGNISSE können zusätzliche Ereignisse aus der Auswahlliste übernommen werden.

7. Die Auswahl kann durch das Setzen von Filtern in der Registerkarte FILTER eingeschränkt werden.
8. Starten Sie die Aufzeichnung im Profiler.
9. Starten Sie den SQL Query Analyzer und geben Sie einige SQL-Anweisungen ein.
10. Die Ausführung der SQL-Anweisungen wird im Profiler, wie im Bild 8.2 zu sehen, registriert.

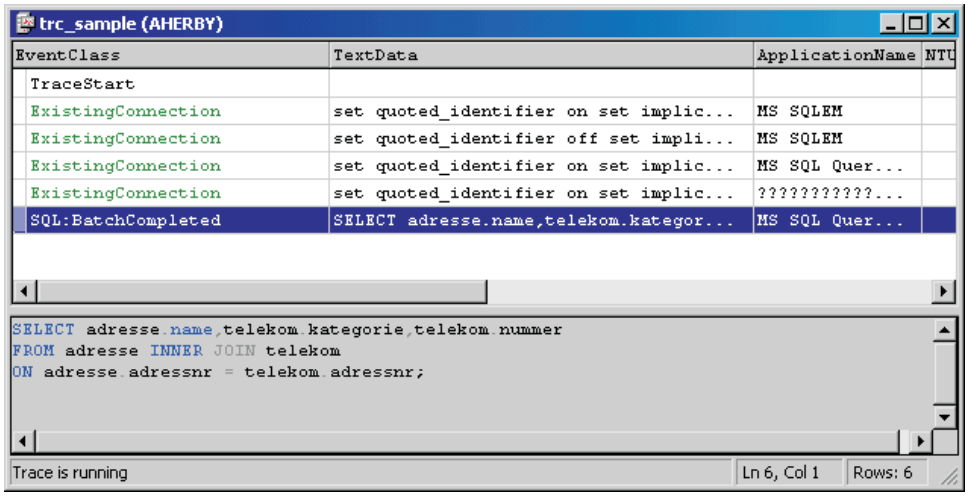


Abbildung 8.2: Ablaufverfolgung mit dem SQL Server Profiler

11. Beenden Sie die Ablaufverfolgung.

Indexoptimierungs-Assistent

Der *Indexoptimierungs-Assistent* gibt eine Empfehlung aufgrund der Auswertung der *Ablaufverfolgungsdatei*, welche Indizes vorhanden sein sollten.

1. Starten Sie den Indexoptimierungs-Assistent im Profiler über WERKZEUGE/ INDEXOPTIMIERUNGS-ASSISTENT.
2. Klicken Sie im WILLKOMMENSDIALOGFELD auf WEITER.
3. Im nächsten Dialogfeld geben Sie die gewünschte Datenbank an (siehe Abbildung 8.3).
4. Die aufgezeichnete Ablaufverfolgung muss im nächsten Dialogfeld gemäß der Abbildung 8.4 spezifiziert werden.

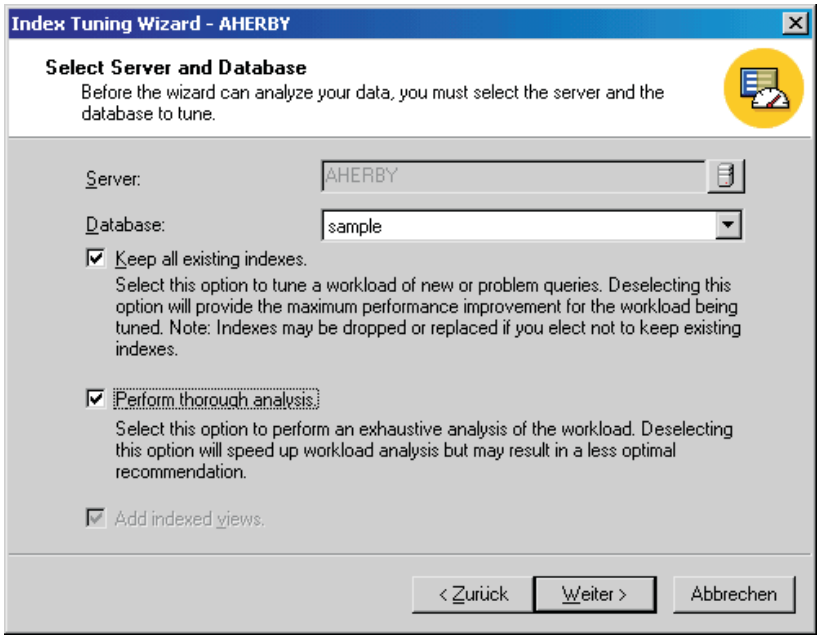


Abbildung 8.3: Angabe der Datenbank im Indexoptimierungs-Assistent

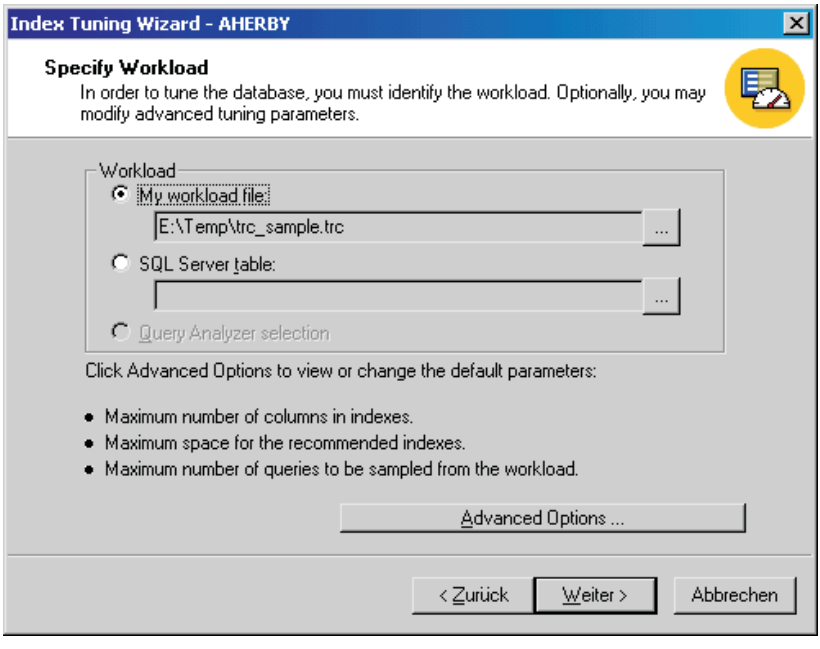


Abbildung 8.4: Angaben zur Ablaufverfolgung

5. Wählen Sie anschließend die Tabellen, für die eine Analyse vorgenommen werden soll.
6. Wenn Sie auf WEITER klicken, wird die Analyse durchgeführt.
7. Das Ergebnis kann in verschieden Berichten aufgerufen werden.

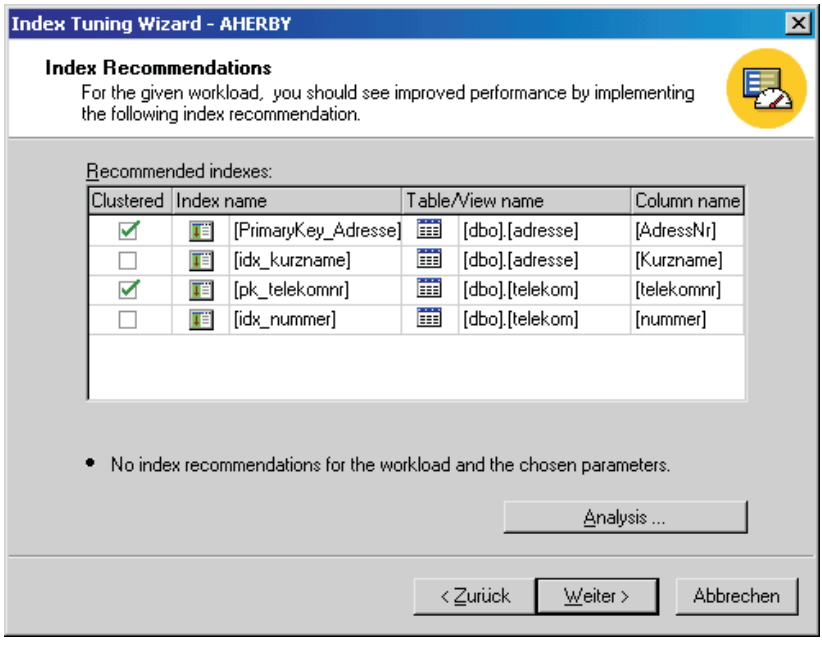


Abbildung 8.5: Auswertung mit dem Indexoptimierungs-Assistent

Eine *Empfehlung* besteht aus SQL-Anweisungen, die ausgeführt werden können, um neue, effizientere Indizes zu erstellen und gegebenenfalls vorhandene Indizes zu löschen, die als *ineffizient* bewertet wurden. Indizierte Sichten werden auf Plattformen empfohlen, die die Verwendung dieser Sichten unterstützen. Nachdem der Indexoptimierungs-Assistent eine Empfehlung vorgeschlagen hat, stehen für die Empfehlung die folgenden Optionen zur Verfügung:

- Sofortige Implementierung.
- Planen der Implementierung zu einem späteren Termin, indem ein *SQL Server-Auftrag* erstellt wird, der ein SQL-Skript ausführt.
- Speichern der Empfehlung in einem SQL-Skript, so dass sie vom Benutzer zu einem späteren Zeitpunkt oder auf einem anderen Server manuell ausgeführt werden kann.

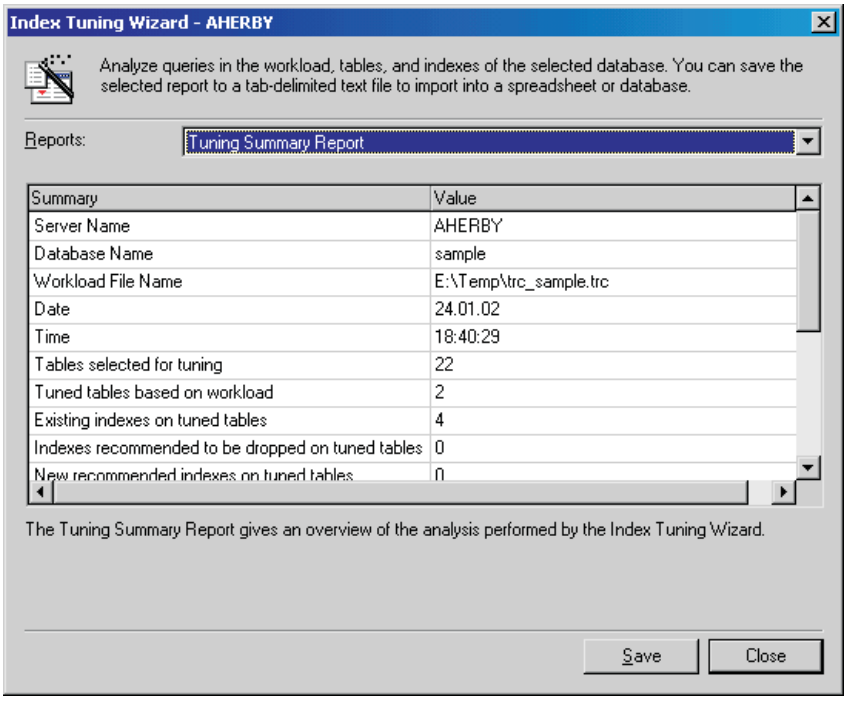


Abbildung 8.6: Empfehlungen des Assistenten

Der Indexoptimierungs-Assistent schließt alle Index- oder Abfragehinweise in die endgültige Empfehlung ein, selbst wenn der Index für die Tabelle nicht optimal geeignet ist. Indizes in anderen Tabellen, auf die in der Abfrage verwiesen wird, werden möglicherweise vorgeschlagen und empfohlen. Alle als Hinweise angegebenen Indizes sind jedoch stets Teil der endgültigen Empfehlung. Die Hinweise verhindern möglicherweise, dass der Indexoptimierungs-Assistent einen besseren Ausführungsplan wählt. Entfernen Sie alle Indexhinweise aus Abfragen, bevor Sie die Arbeitsauslastung analysieren.

8.1.2 Empfehlungen für den optimalen Einsatz von Indizes

Die folgenden Empfehlungen können für das Erstellen von Indizes gegeben werden:

1. Schreiben Sie Abfragen, die möglichst viele Zeilen in einer *einzigen Anweisung* aktualisieren, anstatt hierfür mehrere Abfragen zu verwenden. Wenn nur eine Anweisung verwendet wird, kann der Index optimal verwaltet werden.
2. Verwenden Sie die Werkzeuge des DBMS, um die Abfragen zu *analysieren* und Indexempfehlungen zu geben.

3. Verwenden Sie für *gruppierte Indizes Schlüssel aus ganzen Zahlen*. Gruppierte Indizes bieten darüber hinaus den Vorteil, dass sie für eindeutige, Nicht-NULL- oder IDENTITY-Spalten erstellt werden. Weitere Informationen finden Sie unter »Verwenden gruppierter Indizes«.
4. Erstellen Sie *nicht gruppierte Indizes* für alle Spalten, die häufig in Abfragen verwendet werden. Dies kann den Nutzen von abgedeckten Abfragen maximieren. Weitere Informationen finden Sie unter »Verwenden nicht gruppierter Indizes«.
5. Überprüfen Sie die *Datenverteilung* in indizierten Spalten. Häufig dauert eine Abfrage deshalb sehr lange, weil eine indizierte Spalte mit wenigen eindeutigen Werten verwendet wird oder weil sie eine Verknüpfung mit einer solchen Spalte durchführt.

8.2 Transaktionen

Den Begriff »Transaktionen« kennen wir von unserer Bank. Wir bringen unser »sauer Verdientes« zu der Bank unseres Vertrauens und dort liegt es nicht einfach herum, sondern das Geld wird an andere verliehen. Diese Bankgeschäfte nennt man auch Transaktionen. So oder so ähnlich, wurden wir alle in die Welt der Geldgeschäfte eingeweiht. Wir wollen beim Thema bleiben und ein Fallbeispiel aus dem Bankgeschäft soll uns den Begriff Transaktion im Zusammenhang mit der Datenbanktechnologie näher bringen.

Beispiel 8.1:

Ein Kunde bringt Fremdwährung, die er noch aus dem vergangenen Urlaub hat, in die Filiale seiner Hausbank. Der umgerechnete Betrag, den er durch Einzahlung der Fremdwährung gutgeschrieben bekommt, soll auf sein Konto eingezahlt werden. Es handelt sich um einen Betrag in Höhe von 234,45 Euro. Gleichzeitig benötigt er Bargeld in Höhe von 150,00 Euro. Der Sachbearbeiter an der Bank gibt über seine Eingabemaske beide Vorgänge ein. Zunächst wird der Betrag der Fremdwährung dem Kunden auf sein Konto gutgeschrieben. Dabei wird der entsprechende Datensatz in eine Tabelle eingefügt. In diesem Moment gibt es einen Systemausfall. Die weiteren Aktionen können nicht mehr ausgeführt werden. Das würde bedeuten der Kontostand des Kunden wäre nach erneutem Systemstart nicht korrekt. So lang der Kunde dabei Gewinn macht, wird er kein größeres Problem damit haben. Doch für denjenigen, der dabei finanzielle Verluste hat, wird sich der Spaß vermutlich in Grenzen halten.

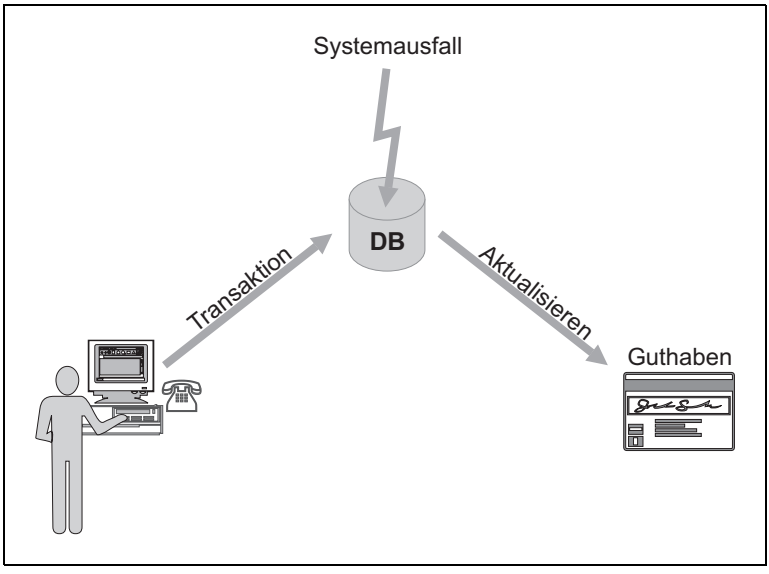


Abbildung 8.7: Systemausfall während einer Transaktion

Damit solche Dinge in der Praxis nicht passieren, gibt es gewisse Anforderungen an Aktionen, die an einer Datenbank ausgeführt werden. Aktionen, die als logische Einheit betrachtet werden können, nennt man *Transaktionen*. Die erste Bedingung, die eine Transaktion erfüllen muss, kann aus dem Beispiel 8.1 abgeleitet werden. Alle Aufgaben, die innerhalb einer Transaktion formuliert sind, müssen ausgeführt werden oder keine. Andernfalls würde es verheerende Folgen haben. Wenn die eingegeben Kontonummer am Bankautomat falsch ist, dann muss der gesamte Vorgang abgebrochen werden.

Für Transaktionen und ihre Verarbeitung gelten die folgende vier Regeln:

1. Sie sind *atomar*, werden also ganz oder gar nicht ausgeführt.
2. Sie stellen in sich *konsistente Verarbeitungseinheiten* dar und lassen eine Datenbasis in einem konsistenten Zustand zurück.
3. Sie laufen quasi *isoliert* ab, d.h. sie bekommen die Wirkung paralleler Transaktionen nicht zu spüren und haben auch keinen Einfluss auf diese.
4. *Dauerhaftigkeit*: Erfolgreich abgeschlossene Transaktionen werden durchgesetzt, d.h. auf dem Plattenspeicher »verewigt«.

Belegt man jede Regel im Englischen mit einem Schlagwort, ergibt dies das allgemein bekannte Merkwort *ACID*.

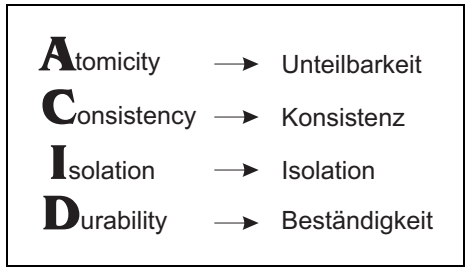


Abbildung 8.8: Eigenschaften einer Transaktion

8.2.1 Arbeiten mit Transaktionen

Eine Transaktion kann einfach als eine Folge von Lese- und Schreibzugriffen auf die Datenbasis verstanden werden. Wie im Beispiel 8.1 gezeigt, machen durchzuführende Operationen nur Sinn, wenn sie alle und in der richtigen Reihenfolge durchgeführt werden. Was zwischen »begin of transaction« (`BOT`) und `COMMIT` (Festschreiben) steht, ist eine unteilbare Einheit. Eine SQL-Transaktion wird initiiert, wenn der relvante *SQL-Agent* eine »transaktionsinitiierende« SQL-Anweisung ausführt und der SQL-Agent nicht bereits eine Transaktion bearbeitet. Nach einer nur teilweisen Durchführung wäre die Datenbasis inkonsistent, d.h. in einem inkorrekten Zustand.

Es kann aber trotzdem vorkommen, dass eine erst teilweise ausgeführte Transaktion abgebrochen werden muss (*ROLLBACK*). Dabei müssen sämtliche Spuren einer abgebrochenen Transaktion wieder beseitigt werden. Bei Beendigung einer Transaktion mit `COMMIT` werden alle von der Transaktion vorgenommenen Änderungen für andere Transaktionen sichtbar gemacht. Die Beendigung mit *ROLLBACK* sorgt dafür, dass alle von der Transaktion vorgenommenen Änderungen zurückgenommen werden; diese Änderungen werden nie für andere Transaktionen sichtbar.

Beispiel 8.2:

```
BEGIN TRANSACTION
INSERT INTO rechnung VALUES (1, 'HHKU0002', NULL, NULL, 35, 20, 440, 182.40,
1322.4, NULL, NULL, NULL, NULL)
INSERT INTO rechnung VALUES (2, 'HHKU0002', NULL, NULL, 35, 40, 50, 232, 1682,
NULL, NULL, NULL, NULL)
INSERT INTO rechnung VALUES (3, 'BEKL0001', NULL, NULL, 50, 16, 50, 136, 986,
NULL, NULL, NULL, NULL)
IF @@error > 0
     ROLLBACK TRANSACTION
ELSE
     COMMIT TRANSACTION
GO
```

Das Beispiel 8.2 zeigt, wie in einer Transaktion drei Rechnungsdatensätze in die Tabelle *rechnung* eingefügt werden. Die Transaktion beginnt mit den Schlüsselwörtern `BEGIN TRANSACTION`. Es folgen die SQL-Anweisungen, die die erforderlichen Aktionen durchführen. IM IF ELSE-Konstrukt erfolgt eine Fehlerabfrage. Die Funktion @@ERROR gibt einen Wert größer Null zurück, wenn ein Fehler in den vorangestellten SQL-Anweisungen ein Fehler aufgetreten ist. In diesem Fall wird ein ROLLBACK ausgeführt. Das heißt, die Transaktion wird nicht ausgeführt und alle Änderungen werden rückgängig gemacht. Andernfalls wird die Transaktion mit einem `COMMIT` abgeschlossen und die Datensätze werden eingefügt.

In Datenbanksystemen geht es also hauptsächlich um Transaktionsverarbeitung! Immer wenn vom Client eine Anfrage an den Server gerichtet wird, haben wir es mit einer Transaktion zu tun.

8.2.2 Transaktionsprotokolle

Änderungen an der Datenbank können in einem *Transaktionsprotokoll* aufgezeichnet werden. Aktionen wie `INSERT`, `UPDATE`, `DELETE` oder sicherheitsrelevante Änderungen an der Datenbank, die als SQL-Anweisungen in einer Transaktion abgearbeitet werden, werden protokolliert. Zu jeder Datenbank gehört mindestens ein Transaktionsprotokoll, das automatisch beim Anlegen einer neuen Datenbank erstellt wird und dessen maximale Größe der Besitzer der Datenbank definieren kann. Außerdem ist es möglich das Protokoll an einem bestimmten Punkt abzuschneiden, damit es nicht irgendwann den gesamten Festplattenplatz für sich beansprucht.

Für jeden Transaktionsschritt wird ein Eintrag in die Protokolldatei gemacht; diese Einträge kommen in einen *Pufferbereich* im Hauptspeicher und von dort in einen separaten perma-nenten Speicher, der so ausfallsicher wie möglich ist. Jeder Eintrag enthält die Information, welche Transaktion, welches Datum auf welcher Seite wie verändert hat. Besonders wichtig ist ein geeignetes Protokollverfahren. Beim *Write-Ahead-Protokoll* werden zunächst alle Änderungen an der Datenbank ins Protokoll geschrieben und dann erst bei fehlerfreier Ausführung der SQL-Anweisungen werden die Änderungen dauerhaft in der Datenbank übernommen. Bei Fehlern kann das DBMS anhand des Transaktionsprotokolls die Datenbank wieder vollständig herstellen. Dabei werden Transaktionen, die im Protokoll stehen aber noch nicht ausgeführt wurden, überprüft und ggf. durchgeführt.

8.2.3 Paralleles Verarbeiten von Transaktionen

Bei einigen DBMS ist es möglich, über Einstellungen in den Datenbankoptionen einen Modus zu erzwingen, bei dem seriell Transaktionen von nur einem Benutzer verarbeitet werden. Dies ist jedoch nur in einer Entwicklungs- oder Testumgebung sinnvoll. Große Datenbanksysteme müssen laufend Transaktionen verarbeiten und zwar viele

»gleichzeitig« und so schnell wie möglich. Nun sollte man wissen: Transaktionen beschäftigen das System auf verschiedene Weise:

Es müssen Daten von Platte in den Hauptspeicher gelesen, oder in umgekehrter Richtung zurückgeschrieben werden. Es wird Rechenleistung benötigt, um die Daten zu verarbeiten.

Plattenzugriffe sind deutlich langsamer, als das Rechnen in der CPU. Die Zugriffe auf die Datenbasis müssen also beschleunigt werden. Das geschieht durch nebenläufige Verarbeitung der Operationen verschiedener Transaktionen. Man wendet eine Methode an, die bei laufendem Betrieb automatisch konfliktserialisierbare Ablaufpläne erzeugt, z.B. ein *sperrbasiertes* Protokoll. Es beruht auf Sperren der Datenbankobjekte. Eine optimale Leistung wird erreicht, wenn die Anzahl der Sperren auf die Datenmenge, die jede Sperre hält, abgestimmt ist. Es können verschiedene Arten von Ressourcen gesperrt werden (z.B. Zeilen, Datenseite, Tabelle oder Datenbank). Sperren auf Ebene der Datenbank würde eine Parallelität der Transaktion verhindern. Je weiter man in dieser *Sperrhierachie* geht, desto höher wird der Verwaltungsaufwand. Bei einer Sperre auf Zeilenebene muss eine Vielzahl an Prüfungen ablaufen, die festzustellen haben, ob die Zeilen nicht bereits von anderen Transaktionen gesperrt sind. Gleichwohl wäre bei einer Sperre auf Zeilenebene die *Parallelität* am größten.

Sperren können folgende Situationen verhindern, welche die Integrität gefährdet:

Abhängigkeit von Daten, für die kein COMMIT ausgeführt wurde (Dirty Read)

Eine solche Abhängigkeit kommt vor, wenn eine Transaktion Daten liest, die von einer anderen geändert wurden, diese Transaktion aber nicht korrekt abgeschlossen wurde. Die Transaktion nimmt unter Umständen Änderungen an Daten vor, die unstimmig oder nicht vorhanden sind (siehe Abbildung 8.9).

Verlorene Aktualisierung

Eine Aktualisierung kann verloren gehen, wenn eine Transaktion die Änderungen einer anderen überschreibt. Zwei Benutzer ändern die gleichen Daten und nur die Änderung der letzten Transaktion wird gespeichert.

Inkonsistente Analyse

Eine Transaktion liest mehrmals den gleichen Datensatz. Zwischen den Lesevorgängen wird der Datensatz von einer anderen Transaktion verändert. Jeder Lesevorgang führt zu inkonsistente Daten, da die Lesevorgänge innerhalb einer Transaktion stattfinden.

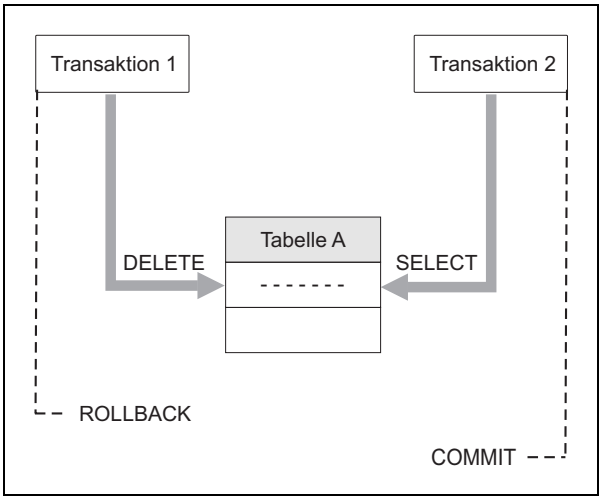


Abbildung 8.9: Dirty Read durch Transaktionen

Phantomzeilen

Eine Transaktion liest Zeilen, die von einer anderen Transaktion während der Ausführung hinzugefügt wurde. *Phantome* können auftreten, wenn Transaktionen nicht vollständig voneinander getrennt sind. Transaktion T1 liest z.B. Zeilen aus einer Tabelle Tab1, die eine Bedingung erfüllen. Transaktion T2 fügt der Tabelle Tab1 eine weitere Zeile hinzu, die der Bedingung aus der Transaktion T1 entsprechen. In der Transaktion T1 wird erneut die Tabelle Tab1 mit der gleichen Bedingung abgefragt. Nun wird eine zusätzliche (Phantom-) Zeile gelesen.

Der *Scheduler*, eine Systemkomponente die wir bereits bei der Datenbankarchitektur kennen gelernt haben, setzt temporäre Sperren; es gibt zwei Modi:

- S-Sperren (»S« für »shared«)
- X-Sperren (»X« für »exclusive«)

Eine Transaktion darf ein Datenbankobjekt lesen, wenn sie eine *S-Sperre* darauf erhalten hat und schreiben, wenn sie eine *X-Sperre* darauf erhalten hat. Dabei gilt, S-Sperren sind nur erhältlich auf Objekte, die noch nicht X-gesperrt sind und diese wiederum sind nur erhältlich auf Objekte, die weder S- noch X-gesperrt sind.

Das strikte 2-Phasen-Sperrprotokoll

Es funktioniert, wie eben beschrieben. Das Besondere bei dieser Art Sperrprotokoll ist, dass jede Transaktion die Sperren, die sie in ihrem Verlauf bekommen hat, bis zum Ausführungsende behält. Die Transaktion beansprucht dadurch zwar Ressourcen, die von anderen Transaktionen gebraucht werden könnten, aber dafür sorgt das *strenge*

2-Phasen-Sperrprotokoll für *Serialisierbarkeit* und zusätzlich für die *Rücksetzbarkeit* abzubrechender Transaktionen.

Bei diesem Protokoll kann es aber vorkommen, dass eine Transaktion eine angeforderte Sperre im Moment nicht erhalten kann, dann kommt sie in eine Warteschlange und muss dort auf den Erhalt »ihrer« Sperren warten. Damit ist noch nicht alles geregelt, denn auch beim strengen 2-Phasen-Sperrprotokoll ist man noch nicht alle Probleme los, wie das folgende Beispiel zeigt.

Beispiel 8.3:

Eine Transaktion T1 hat die Tabelle A mit einer Sperre belegt. Gleichzeitig hat Transaktion T2 eine Sperre für Tabelle B eingerichtet. Jede Transaktion versucht nun auf das von der anderen Transaktion gesperrte Objekt zuzugreifen. So verhindern beide Transaktionen, dass die andere die Verarbeitung fortsetzt. Eine solche Blockade nennt man *Deadlock*. Was kann man gegen Deadlocks tun?

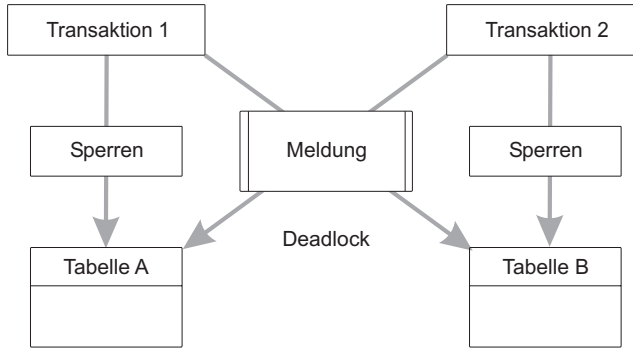


Abbildung 8.10: Deadlock, verursacht durch zwei Transaktionen

Es gibt einen *pessimistischen* und einen *optimistischen* Ansatz. Pessimistisch ist, durch einen Algorithmus dafür zu sorgen, dass Blockaden gar nicht erst entstehen. Eine pessimistische Vorgehensweise ist auch das *Zeitstempelverfahren*. Der Zugriff auf Objekte erfolgt im Konfliktfall in der Reihenfolge des Startzeitpunktes der Transaktion. Der optimistische Ansatz geht davon aus, dass solche Situationen selten sind und lässt sie deshalb vorkommen, es wird aber ein *Wartegraph* aufgebaut und laufend kontrolliert. Das Risiko eines *Deadlocks* kann auch schon durch die Beachtung einiger Richtlinien beim Arbeiten mit Transaktionen verringert werden:

- Verkürzung der Transaktion, indem die Anzahl der Schritte gering gehalten wird.
- Verkürzung der Transaktionszeit, indem komplexe Abfragen vermieden werden.
- Verwenden der Ressourcen in allen Transaktionen in gleicher Reihenfolge.

8.3 Sperren von Ressourcen

Dass Ressourcen gesperrt werden, haben wir bereits im vorherigen Abschnitt festgestellt. Normalerweise werden die Sperren vom DBMS gesetzt und auch wieder aufgelöst. Man ist dem »Schalten und Walten« des DBMS jedoch nicht völlig hilflos ausgeliefert. Auf zwei Ebenen hat der Administrator oder Programmierer die Möglichkeit, in den *Sperrmechanismus* des DBMS einzugreifen:

- Sperroptionen auf Sitzungsebene,
- Sperroptionen auf Tabellenebene.

8.3.1 Sperroptionen auf Sitzungsebene

Das Sperrverhalten kann auf Sitzungsebene durch Festlegung der *Transaktionsisolationsstufe* erfolgen. Die Isolationsstufe definiert, wie Transaktionen auf die Ressourcen zugreifen dürfen. Eine Isolationsstufe kann eine Transkation vor unerwünschten Einflüssen anderer Transaktionen schützen. Eine definierte Isolationsstufe ist für die Dauer einer Sitzung gültig. Sie wird jedoch durch Sperrspezifikationen in einzelnen SQL-Anweisungen außer Kraft gesetzt. Die Tabelle 8.1 beschreibt die verschiedenen Isolationsstufen.

Isolationsstufe	**Beschreibung**
Read Uncommitted	gibt an, dass gemeinsame Sperren aufrechterhalten werden, während die Daten gelesen werden. Die Daten können jedoch vor dem Ende der Transaktion geändert werden.
Read Committed	Das DBMS wird angewiesen, keine gemeinsamen Sperren auszugeben und exklusive Sperren nicht anzuerkennen.
Repeatable Read	Sperren werden für alle Daten eingerichtet, die in einer Abfrage verwendet werden; sie verhindern, dass andere Benutzer die Daten aktualisieren. Lesesperren werden bis zum Ende der Transaktion aufrechterhalten.
Serializable	verhindert, dass andere Benutzer neue Zeilen aktualisieren oder einfügen, die mit den Kriterien der WHERE-Klausel der Transaktion übereinstimmen. Von den 4 Isolationsstufen ist diese die restriktivste.

Tabelle 8.1: Isolationsstufen bei der Transaktionsverarbeitung

Mit der SQL-Anweisung `SET TRANSACTION ISOLATION LEVEL` kann die Isolationsstufe festgelegt werden.

Syntax:

```
SET TRANSACTION ISOLATION LEVEL
    {   READ COMMITTED |
        READ UNCOMMITTED |
        REPEATABLE READ |
        SERIALIZABLE
    }
```

Beispiel 8.4:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

Die Angabe der Isolationsstufe beeinflusst auch Transaktionsunsicherheiten, die im Abschnitt 8.2.3 beschrieben wurden. Die Tabelle 8.2 zeigt auf, welche Verletzungen bei den jeweiligen Isolationsstufen auftreten können.

Isolationsstufe	Dirty Read	Inkonsistente Analyse	Phantom
Read Uncommitted	Ja	Ja	Ja
Read Committed	Nein	Ja	Ja
Repeatable Read	Nein	Nein	Ja
Serializable	Nein	Nein	Nein

Tabelle 8.2: Transaktionsunsicherheiten in Abhängigkeit der Isolationsstufe

Die Isolationsstufe READ COMMITTED ist üblicherweise die Standardeinstellung.

8.3.2 Sperroptionen auf Tabellenebene

Auf Ebene der Tabelle können mittels SQL-Anweisungen *Sperrhinweise* mitgegeben werden. Dadurch wird der zu verwendende Sperrtyp festgelegt. Sperrhinweise auf Tabellenebene können verwendet werden, wenn eine präzisere Steuerung der *Sperrtypen* für ein Objekt notwendig wird. Diese Sperrhinweise setzen die aktuelle Isolationsstufe für Transaktionen dieser Sitzung außer Kraft.

Die möglichen Sperroptionen für den SQL Server können sie der folgenden Tabelle entnehmen:

Sperroption	Beschreibung
HOLDLOCK	Sperre wird bis zum Ende der Transaktion aufrechterhalten.
NOLOCK	gibt keine gemeinsamen Sperren aus und erkennt exklusive Sperren nicht an.

Tabelle 8.3: Sperroptionen beim SQL Server

Sperroption	Beschreibung
PAGLOCK	setzt Seitensperren in solchen Fällen ein, in denen in der Regel eine einzelne Tabelle gesperrt wird.
READCOMMITTED	gleiche Sperrverhalten wie bei der gleichnamigen Isolationsstufe.
READPAST	Bei dieser Option überspringt eine Transaktion alle Zeilen, die von anderen Transaktionen gesperrt sind und die normalerweise in der Ergebnismenge angezeigt würden.
READUNCOMMITTED	entspricht der Option NOLOCK.
REPEATABLEREAD	entspricht der Sperrcharakteristik der gleichnamigen Isolationsstufe.
ROWLOCK	Sperren auf Zeilenebene werden verwendet.
SERIALIZABLE	entspricht der Sperrcharakteristik der gleichnamigen Isolationsstufe.
TABLOCK	verwendet eine Tabellensperre anstatt der differenzierteren Sperren auf Zeilen- oder Seitenebene.
TABLOCKX	Eine Tabelle wird mit einer exklusiven Sperre belegt.
UPDLOCK	verwendet Aktualisierungssperren anstelle von gemeinsamen Sperren.
XLOCK	verwendet eine exklusive Sperre, die bis zum Ende der Transaktion für alle von der Anweisung verarbeiteten Daten aufrechterhalten wird.

Tabelle 8.3: Sperroptionen beim SQL Server (Forts.)

Angenommen die Firmenadministration hat neue Bewertungsschlüssel für die Mitarbeiter beschlossen. Die Schlüssel sind in der Tabelle *kategorie* definiert. Die Tabelle soll nun aktualisiert werden. Damit während des Aktualisierungsvorgangs kein Benutzer auf die Tabelle *kategorie* zugreift, wird die Transaktion mit einer exklusiven Tabellensperre belegt.

Beispiel 8.5:

```
BEGIN TRANSACTION
UPDATE kategorie WITH (TABLOCKX)
SET name = CONVERT(nvarchar, (CONVERT(real, name) + 2.4))
WHERE gruppe = 'BEWSL'
COMMIT TRANSACTION
```

Die Sperroption wird direkt nach der Tabellenangabe in Verbindung mit der WITH-Klausel definiert. Weil in einer Spalte mit dem Datentypen `nvarchar` eine Addition durchgeführt werden soll, muss der Wert zunächst in den Datentyp `real` und anschließend wieder in den Datentyp `nvarchar` konvertiert werden. Dies geschieht in dem angegebenen Beispiel mit der CONVERT-Funktion.

Auch beim DBMS PostgreSQL kann auf Tabellenebene eine Sperroption festgelegt werden. Die Steuerungsanweisung und auch die möglichen Optionen weichen von der beim SQL Server ab. Die Sperroption wird mit der LOCK-Anweisung definiert.

Syntax bei **PostgreSQL**:

```
LOCK [ TABLE ] name IN option
```

Beschreibung:

name
Name einer existierenden Tabelle, die gesperrt werden soll.

option
Eine der Sperroptionen aus Tabelle 8.4.

Die Sperroptionen, die bei PostgreSQL möglich sind, werden in der folgenden Tabelle aufgeführt.

Sperrmodus	**Beschreibung**
ACCESS SHARE MODE	schützt die Tabelle vor Modifizierung durch ALTER TABLE, DROP TABLE und VACUUM.
ROW SHARE MODE	gemeinsam genutzte Sperre auf Zeilenebene.
ROW EXCLUSIVE MODE	exklusive Sperre auf Zeilenebene.
SHARE MODE	wird automatisch bei CREATE INDEX aktiviert. Die Tabelle wird vor gleichzeitigen Veränderungen geschützt.
EXCLUSIVE MODE	Es werden alle gleichzeitigen ROW SHARE/ SELECT...FOR UPDATE Abfragen verhindert.
SHARE ROW EXCLUSIVE MODE	gleiche Verhalten wie beim EXCLUCIVE MODE, erlaubt jedoch zusätzlich eine Sperre auf Zeilenebene durch andere.
ACCESS EXCLUSIVE MODE	Durch Sperren werden alle gleichzeitigen Aktionen an einer Tabelle verhindert. Dieser Modus ist der restriktivste.

Tabelle 8.4: Sperroptionen beim PostgreSQL

Beispiel 8.6:

```
BEGIN WORK;
LOCK TABLE kategorie IN SHARE ROW EXCLUSIVE MODE;
DELETE FROM mitarbeiter WHERE qualifikation IN
    (SELECT kategorie FROM kategorie WHERE name = 'Qu');
DELETE FROM kategorie WHERE name = 'Qu';
COMMIT WORK;
```

In dem Beispiel 8.6 werden Datensätze der Tabelle *mitarbeiter* und *kategorie* in einer Transaktion gelöscht. Um einen Deadlock zu verhindern, wird die Tabelle im SHARE ROW EXCLUSIVE MODUS gesperrt.

8.4 Arbeiten mit verteilten Daten

Das relationale Datenmodell war und ist so erfolgreich, weil es sich sehr gut zum Beschreiben vieler realer und besonders der betriebswirtschaftlichen Zusammenhänge eignet. Auch das Arbeiten mit verteilten Daten und die Vorteile die sich daraus ergeben, sind durch betriebswirtschaftliche (oder standortabhängige) Gegebenheiten begründet. Unternehmen sind häufig auf mehrere Niederlassungen aufgeteilt. Meist existieren mehrere Fabriken, Filialen oder Werke. Selbst in einem Werk gibt es verschiedene Abteilungen, Projektgruppen und Bereiche, die eine Verteilung der anfallenden Daten bewirken. Es stellt sich dann die Frage, ob es überhaupt notwendig ist, dass alle Daten zentralisiert werden, d.h. die Daten auf einem zentralen Server zusammengefasst werden.

8.4.1 Pro und Contra

Die verteilte Datenhaltung ist eine Methode, die sinnvoll sein kann, aber nicht in jedem Fall dem System nutzt. Die folgende Gegenüberstellung der Vor- und Nachteile sollen bei einer Entscheidung Hilfestellung geben, ob man mit verteilten Datenbanken arbeiten soll oder nicht.

Vorteile	**Nachteile**
Entspricht den oftmals gegebenen Unternehmensstrukturen.	Es werden neue (komplexe) Algorithmen für die Datenverwaltung benötigt.
Zentrale Verwaltungsaufgaben werden reduziert.	Eine absolute Fehlerfreiheit beim Transaktionsbetrieb kann nicht garantiert werden.
Schnellerer Zugriff auf lokale Daten.	Zusätzliche Verwaltungsaufgaben erfordern zusätzliche Netzwerkaktivitäten.
Ausfallsicherheit.	Es sind leistungsstarke DBMS und Rechnerplattformen notwendig.
Gleichmäßigere Auslastung des Netzwerkes.	

Tabelle 8.5: Vor- und Nachteile für eine verteilte Datenhaltung

8.4.2 Richtlinien für die verteilte Datenhaltung

Damit ein verteiltes System sich gegenüber dem Anwender genauso verhält wie ein nicht verteiltes System, müssen einige Voraussetzungen vorab geschaffen werden. *J.F. Date* hat diese in einem Forderungskatalog aufgelistet:

1. Lokale Eigenständigkeit jedes Rechners
   Jeder Rechner im System sollte ein Maximum an Eigenständigkeit beibehalten. Damit ist gemeint, dass lokale Aktionen erfolgreich und unabhängig von anderen Rechnern im System, durchgeführt werden können. Ein daraus ableitbare Forde-

rung ist die lokale Speicherung und Verwaltung von Daten. Dadurch erhöht sich die Ausfallsicherheit.

2. Keine zentrale Verwaltungsinstanz
   Es darf keine zentrale Instanz geben, welche die verteilte Datenbank verwaltet. Das Gesamtsystem soll durch einen solchen Engpass nicht beeinträchtigt werden.

3. Ständige Verfügbarkeit
   Durch die Verteilung administrativer Funktionalität auf mehrere Rechner, sollte es nicht mehr erforderlich sein, das gesamte System herunterzufahren. Ein Teilsystem wird daher immer verfügbar sein.

4. Lokale Unabhängigkeit
   Daten werden für die verschiedenen Anwendungen auf den lokalen Rechner geholt. Anwendungen können deshalb auf jedem Rechner ohne größere Anpassungen laufen.

5. Unabhängigkeit gegenüber Fragmentierung
   Der Begriff der *Fragmentierung* bedeutet in diesem Zusammenhang, dass Dateneinheiten über mehrere Rechner verteilt sein können. Eine große Tabelle kann durchaus auf mehrere Rechner aufgeteilt sein. Dabei müssen jedoch die für den lokalen Standort relevanten Daten auch auf lokalen Rechnern gespeichert werden. Wird eine Kundentabelle beispielsweise auf mehrere Datenbankserver fragmentiert, so muss gewährleistet sein, dass die Kunden, die den einzelnen Niederlassungen eindeutig zugeordnet werden können, dort auch lokal gehalten werden.

6. Unabhängigkeit gegenüber Datenreplikation
   Bei der *Datenreplikation* werden gemeinsam genutzte Daten auf den einzelnen Rechnern in regelmäßigen Abständen synchronisiert. Zwischen den Abgleichterminen kommt es zu Änderungen der Daten auf den einzelnen Rechnern. Außerdem bedeutet es immer einen gewissen Verwaltungsaufwand, die Daten abzugleichen und zu validieren. Das System und die einzelnen Rechner müssen unabhängig von diesen Replikationsvorgängen und -erscheinungen Funktionalität und Datenintegrität gewährleisten.

7. Optimierte verteilte Zugriffe
   Um die Netzwerkbelastung gering zu halten, müssen Zugriffe optimiert werden. Dies bedeutet, dass Daten direkt dort angefordert werden, wo sie auch gehalten werden und nicht über Umwege.

8. Verteilte Transaktionsverwaltung
   Die ACID-Merkmale (siehe Abschnitt 8.2) für die Transaktionen gelten auch für die verteilte Datenhaltung. Transaktionen sind auch hier atomare Einheiten und ihre Ausführung auf den lokalen Rechnern muss global gesteuert werden.

9. Unabhängigkeit von der Hardware
   Komponenten in einem heterogenen Netzwerk müssen miteinander kommunizieren können. Ein UNIX-Rechner muss also auch die Anfragen von einem Windows-Rechner abwickeln können und umgekehrt.

10. Unabhängigkeit von Betriebssystemen
    Das Gleiche, was für die Hardware gilt, muss selbstverständlich auch für die verwendeten Betriebssysteme gelten.

11. Unabhängigkeit vom Netz
    Die notwendigen Netzwerkprotokolle müssen installiert werden, damit es zu einem reibungslosen Datentransfer innerhalb des Netzes (auch bei einem heterogenen Netz) kommen kann.

12. Unabhängigkeit von den Datenverwaltungssystemen
    Auch hier müssen Produkte unterschiedlicher Hersteller zusammenarbeiten können. Durch normierte Datenbankschnittstellen und Datenbankabfragesprache ist dies mit Einschränkungen sicherlich möglich.

Es ist in der Praxis sehr schwierig, alle 12 Punkte gleichzeitig zu erfüllen. So fordert die zweite Regel, es darf keine zentrale Verwaltungsinstanz geben. Dem gegenüber steht die Notwendigkeit einer globalen Kontrolle bei den Transaktionen. Schließlich wird man bemüht sein, das Optimum beim Konzipieren des verteilten Systems herauszuholen und insbesondere das Augenmerk auf hohe Performance, Datenintegrität und Sicherheit legen.

8.4.3 Failover-Clusterunterstützung beim SQL Server

Bei der Methode des *Failoverclusters* werden in einer Gruppe mehrerer Servercomputern Ressourcen, wie z. B. Laufwerke, gemeinsam genutzt. Die einzelnen Server im Cluster werden *Knoten* genannt. Jeder Server bzw. Knoten ist mit dem Netzwerk verbunden, und jeder Knoten kann mit jedem anderen Knoten kommunizieren.

Die gemeinsam genutzten Ressourcen im Cluster werden in *Clustergruppen* zusammengefasst. Wenn ein Failovercluster z. B. über vier Clusterlaufwerke verfügt, können zwei dieser Laufwerke in einer Clustergruppe und die beiden anderen Laufwerke in einer zweiten Clustergruppe zusammengefasst werden. Einer der Knoten im Failovercluster ist der Besitzer der jeweiligen Clustergruppe, obwohl der Besitz zwischen den Knoten übertragen werden kann.

Auf den Knoten im Failovercluster können Anwendungen installiert werden. Bei diesen Anwendungen handelt es sich in der Regel um Serveranwendungen. Die ausführbaren Dateien der Anwendung sowie andere Ressourcen werden in der Regel in einer oder mehreren der Clustergruppen gespeichert, deren Besitzer der Knoten ist. Auf jedem Knoten können mehrere Anwendungen installiert werden. In Bild 8.11 gibt es

vier aktive Knoten mit Lastverteilung. Ein Ausfall von maximal drei Konten wäre möglich, ohne dass das System komplett zusammenbrechen würde.

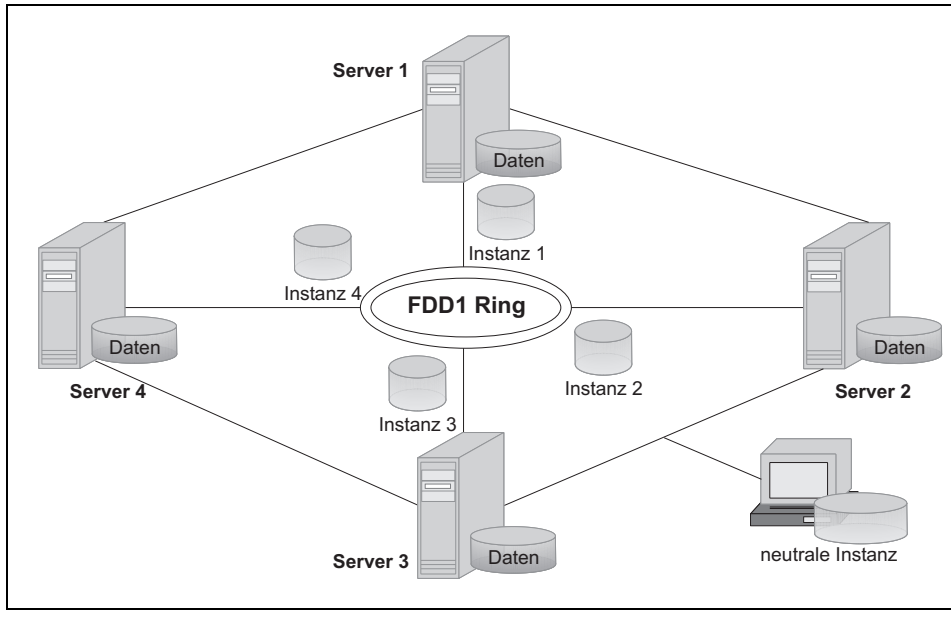


Abbildung 8.11: Beispiel für die Failovercluster-Methode

Die Knoten des Failoverclusters senden sich in regelmäßigen Abständen, so genannte *Statusnachrichten* über das Netz. Wenn die auf allen Knoten installierte Verwaltungssoftware feststellt, dass ein Statussignal von einem der Knoten im Cluster ausgeblieben ist, wird der Server als ausgefallener Server behandelt. Anschließend werden die Clustergruppen und Anwendungsressourcen dieses Knotens automatisch auf die anderen Knoten im Netzwerk übertragen. Der Clusteradministrator gibt die Alternativknoten an, auf die die Clustergruppen übertragen werden, wenn ein bestimmter Knoten ausfällt. Die anderen Knoten setzen das Verarbeiten der, von Benutzern über das Netzwerk gesendeten, Anforderungen für die Anwendungen fort, die vom ausgefallenen Server übertragen wurden.

8.5 Daten importieren und exportieren

Zwei Firmen fusionieren, die EDV wird zentralisiert und von ehemals zwei komplexen Systemen soll schließlich ein DBMS erhalten bleiben, welches die Daten beider Firmen beinhalten soll. Im Klartext heißt dies, die Datenbestände müssen analysiert werden und im einfachsten Fall können sie mit der Importfunktionalität des gewählten DBMS übernommen werden. Schwieriger ist es jedoch, und das ist der Normalfall, wenn die Fremddaten eine völlig andere Struktur aufweisen. Ein anderes durchaus übliches Szenario, ist das Übernehmen von Kundendaten, die als Excel-Tabelle oder einfach nur als ASCII-Datei vorliegen. In diesen und auch in anderen denkbaren Fällen müssen Daten in das DBMS importiert werden. Genauso denkbar wäre aber auch der Wunsch, dass für eine Präsentation ein Ausschnitt aus der Datenbank exportiert werden soll. Das *Importieren* und *Exportieren* von Daten gehört zu den täglichen Anforderungen an Personal und System.

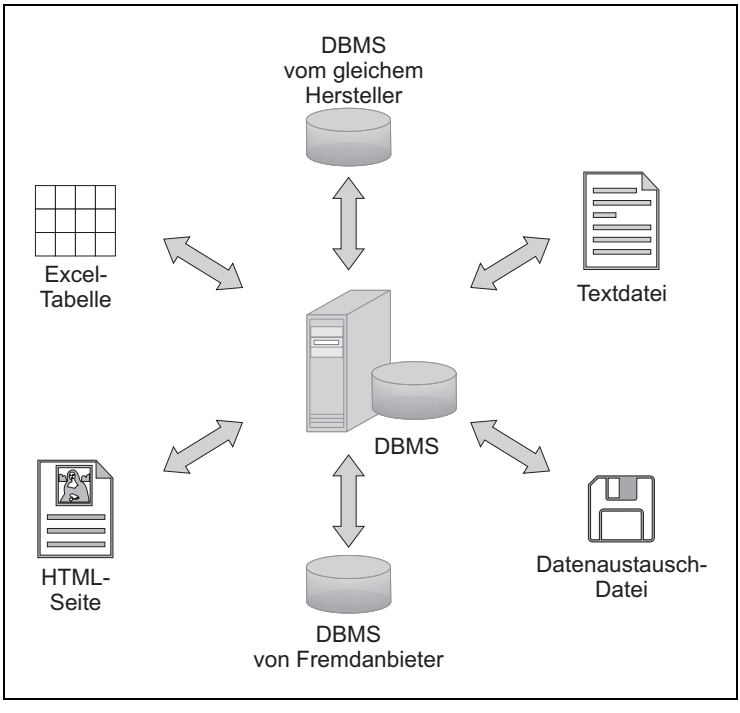


Abbildung 8.12: Import und Export von Daten

8.5.1 Analyse der Fremddaten

Bevor Daten importiert oder exportiert werden können, muss klar sein, welche Struktur das Fremdsystem hat und in welcher Beschaffenheit die Daten vor oder nach dem Import- bzw. Exportvorgang vorliegen. Aufklärung darüber kann am ehesten die Dokumentation oder der Hersteller selbst geben. Manchmal sind die gewonnen Informationen jedoch nicht sonderlich aufschlussreich und man ist gezwungen, selbst die Austauschdaten zu analysieren. Wenn Sie nicht über das System verfügen, welches die Fremddaten erzeugt, so versuchen Sie, die Datei in einem einfachen Editor zu öffnen. Liegen die Rohdaten im ANSI- oder ASCII-Format vor werden Sie keine Schwierigkeiten haben, die Daten in ihr DBMS zu importieren. Für dieses Format gibt es *Import- und Exportfilter* in nahezu jedem DBMS. Allerdings werden die importierten Daten wahrscheinlich nicht Ihrer Datenbankstruktur entsprechen.

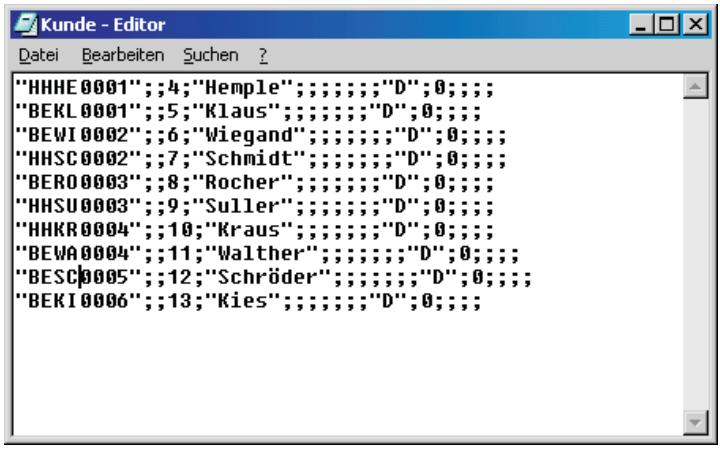


Abbildung 8.13: Access-Exportdatei im Texteditor geöffnet

In der Abbildung 8.13 haben wir eine Exportdatei aus MS Access in einem Texteditor geöffnet. Beim Export wurde bereits das ASCII-Format gewählt, so dass es beim Lesen der Daten keinerlei Schwierigkeiten gibt. Die Datensätze sind zeilenweise aufgelistet und die einzelnen Spalten durch ein Semikolon getrennt. Die Daten dieser Datei könnten ohne weitere Bearbeitung in einem DBMS wie MS SQL Server importiert werden.

Möglich wäre auch ein Dateiinhalt, wie er in Abbildung 8.14 dargestellt ist. Es handelt sich offensichtlich nicht um ein reines ASCII-Format. Die verschiedenen Steuerzeichen, die aus dem Quellsystem stammen, können vom Texteditor nicht interpretiert werden. Im glücklichsten Fall lassen sich zumindest die Rohdaten extrahieren und so über einen Umweg in das eigene DBMS importieren.

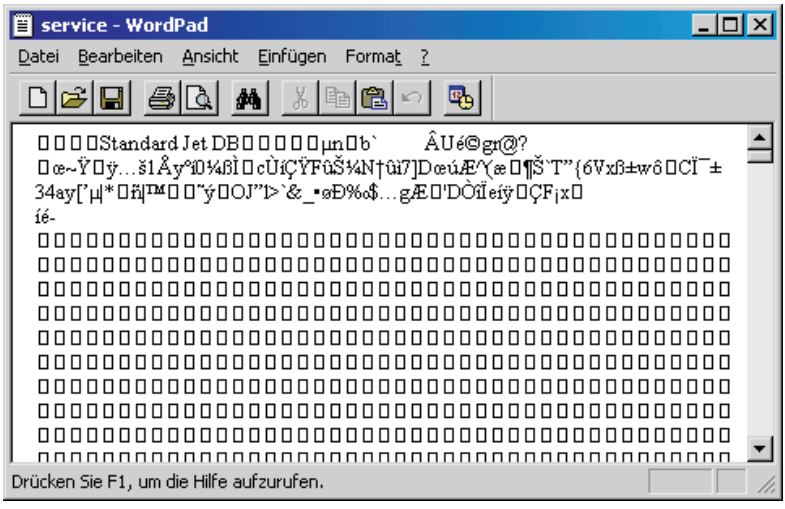


Abbildung 8.14: ASCII-Datei im Texteditor geöffnet

Sind jedoch nicht nur die Informationen des Dateikopfes als Steuerzeichen vorhanden, sondern für die gesamte Datei, wie im Bild 8.15, muss erst eine Dateikonvertierung vorgenommen werden.

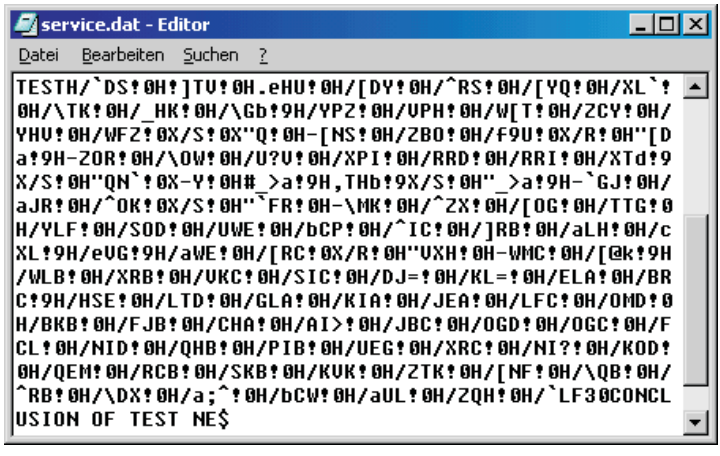


Abbildung 8.15: DAT-Datei im Texteditor geöffnet

Die geöffnete Datei liegt im *DAT-Format* vor. Dieses Format ist besonders bei Banken für den Datenaustausch sehr beliebt. Es ist ein normiertes Format und mit entsprechenden Werkzeugen lassen sich die Rohdaten auch aus einer solchen Datei isolieren.

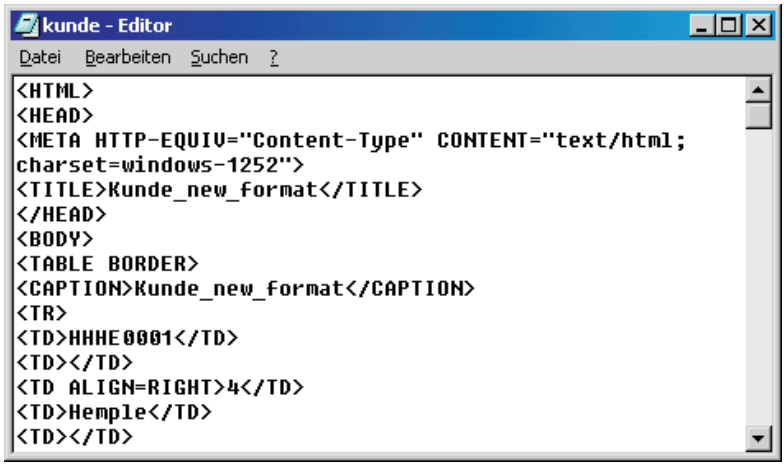


Abbildung 8.16: HTML-Datei im Texteditor geöffnet

Auch HTML-Dateien (siehe Abbildung 8.16) und XML-Dateien werden für den Datenaustausch verwendet. Besonders den XML-Dateien kommt dabei eine besondere Bedeutung zu, die näher im Kapitel 11, wenn es um XML geht, besprochen werden soll.

8.5.2 Auswahl der Werkzeuge

Ist man sich darüber im Klaren, um welches Dateiformat es sich handelt, so müssen entsprechende Werkzeuge bereitgestellt werden. Wie bereits angedeutet, liefern einige DBMS bereits ein Reihe effektiver Werkzeuge, auf die man zurückgreifen kann. Leider hat man nicht immer das Glück, dass für das vorliegende Dateiformat ein *Importfilter* vorliegt. Dann müssen *Konverter* von Drittanbietern herhalten oder man macht sich selbst die Mühe und programmiert ein Konvertierungsprogramm. Die folgende Tabelle gibt eine Übersicht über gebräuchliche Dateiformate und Dienstprogramme zum Konvertieren:

Dateiformat	**Dateinamens-erweiterung**	**Konverter**	**Anbieter**
ASCII	TXT, ASC	GRASS	GRASS GIS
Hypertext Markup Language	HTML	Hyperhelp	Bristol Technology
Datenträgeraustausch	DAT	Datfmt	siehe Begleit-CD
Microsoft Datenbank	MDB	Access Converter	Diamond Edge

Tabelle 8.6: Übersicht über Dateiformate und Konvertierungsprogramme

Dateiformat	Dateinamens-erweiterung	Konverter	Anbieter
Extensible Markup Language	XML	XML2LIT	Mark Lipsman
		XMuLator	Charteris
Rich Text Format	RTF	RTF FormattingKit	Schem GmbH

Tabelle 8.6: Übersicht über Dateiformate und Konvertierungsprogramme (Forts.)

8.5.3 Vorbereiten der Daten

Zum Vorbereiten der Daten gehört, dass sie in einem Format vorliegen, welches ihr DBMS interpretieren kann. Informieren Sie sich über die Import- und Exportfilter ihres DBMS. Fassen Sie als nächstes die Daten logisch so zusammen, wie es das Zielsystem erfordert. Sie haben beispielsweise eine Textdatei, bei der die Kundeninformationen und Informationen zu Aufträgen gemischt in einer Zeile vorkommen. Die Daten aus der Textdatei sollen in ihr DBMS importiert werden, das für diese Informationen zwei Tabellen vorsieht. Vor dem Import müssen nun die Daten entsprechend der Tabellenstruktur in der Datenbank aufgeteilt werden. Bei diesem Vorgang ist es notwendig, Schlüsselfelder und Felder bei denen kein NULL-Wert erlaubt ist, mit Daten zu füllen. Versuchen Sie abzuschätzen, ob es sich lohnt, für diesen Prozess ein kleines Programm zu schreiben. Handelt es sich um einen einmaligen Import mit ca. 100 Datensätzen lohnt sich der Programmieraufwand sicherlich nicht. Werden häufiger Importvorgänge über das gleich Datenaustauschformat erwartet, so sollte eine Schnittstelle für dieses Format programmiert werden, welche die Daten für die Tabellenstruktur vorbereitet. Dadurch wird auch die Wahrscheinlichkeit, dass Daten falsch zugeordnet werden, gering gehalten.

Eine zusätzliche Bedingung ist neben der Tatsache, dass Zieltabellen existieren müssen, welchen die Daten zugeordnet werden können, kompatible *Wertebereiche* für die Spalten. Beachten Sie, dass nicht einfach beliebige Zeichenfolgen in eine Spalte mit dem Datentyp *Integer* eingefügt werden können. Gegebenenfalls müssen Sie die Werte in den einzelnen Spalten nachbearbeiten, damit sie dem Wertebereich des Zielsystems entsprechen.

Eine weitere Folge, die sich aus den unterschiedlichen Wertebereichen ergibt, sind auftretende *Abschneidungen* bei den Daten. Beim Quellsystem wurde für den Kundennamen eine Länge von 40 Zeichen vorgesehen und beim Zielsystem ist dafür nur eine Länge von 20 Zeichen erlaubt. Sind die Namen dann länger als 20 Zeichen, so werden die Zeichen bis zum 20. Zeichen übernommen und die folgenden einfach abgeschnitten. In einem solchen Fall gilt es zu überlegen, ob man mit vorbereitenden Maßnahmen, wie etwa das Aufteilen der Informationen in mehrere Spalten, die Abschneidungen verhindern kann oder ob man sie als Schönheitsfehler in Kauf nimmt.

Die einzelnen Spalten müssen klar definiert sein, d.h. es müssen *Trennzeichen* vorgegeben werden die vom DBMS auch als solche gelesen werden (z.B. Semikolon, Doppelpunkt oder Tabulator). Dabei muss beachtet werden, dass diese Trennzeichen auch in Textfeldern vorkommen können.

Es müssen ferner *Berechtigungen* zum Lesen und Schreiben der Daten vorhanden sein. Informieren Sie sich über die Berechtigungen der Datenbankobjekte.

Zusammenfassend sind die einzelnen Punkte zur Datenvorbereitung hier noch einmal aufgelistet:

- Daten in ein lesbares Format bringen;
- Zieltabellen (oder datenbeschreibende Struktur) müssen existent sein;
- Datenstruktur anpassen:
  - Spaltenanzahl abgleichen,
  - Daten in den Wertebereich des Zielsystems transferieren,
  - Schlüsselfelder mit Daten füllen,
  - Einschränkungen beachten,
  - ggf. Abschneidungen verhindern,
- Trennzeichen für die Spalten definieren;
- Berechtigungen für beteiligte Datenbankobjekte überprüfen.
- Verfügen Sie über MS Access, so können Sie damit einfach über die Zwischenablage Daten spaltenweise anordnen und vorbereiten. Mit Hilfstabellen können Sie dann sehr schnell die Daten in die benötigte Tabellenstruktur bringen. Ein weiterer Vorteil ist die bereits vorhandene Zugriffsmöglichkeit via ODBC auf Fremdsysteme.

8.5.4 Datenimport

Angenommen, die vorbereitenden Schritte sind abgeschlossen und die Daten sind in dem Format, das von den Importmechanismen des DBMS unterstützt wird. Nun steht dem eigentlichen Import fast nichts mehr im Weg. Drei Dinge sollten Sie vorher jedoch noch beachten:

- Deaktivieren Sie Trigger, die durch den Import ausgelöst werden könnten.
- Löschen Sie ggf. Indizes in den Tabellen, in die eine große Anzahl an Datensätze eingefügt wird.
- Versuchen Sie durch den Importvorgang nicht den laufenden Betrieb zu beeinträchtigen.

Die Datei aus Abbildung 8.13 soll im Folgenden mit dem *DTS* (Data Transformation Services) in den SQL Server importiert werden. Sehen Sie dazu die nötigen Schritte:

1. Starten Sie *DTS* im *Enterprise Manager*, indem Sie im Menü WERKZEUGE selektieren und im Kontextmenü von Data Transformation Services IMPORT... anklicken.
2. Es wird der Assistent für den Datenimport und -export gestartet.
3. Wählen Sie bei den Quellangaben *Textdatei*.
4. Geben Sie *Name* und *Verzeichnis* der Importdatei an.
5. Machen Sie Angaben zum *Dateiformat* (siehe Abbildung 8.17). Verwechseln Sie an dieser Stelle nicht den *Spaltendelimeter* mit dem *Zeilendelimeter*.
6. Machen Sie Angaben zur *Zieldatenbank*.
7. Wählen Sie die *Tabelle*, in die importiert werden soll.
8. Geben Sie den *Zeitpunkt* für den Import an oder importieren Sie direkt nach Fertigstellung des Importauftrages.
9. Beenden Sie den Vorgang durch Klicken auf die Schalfläche FERTIG STELLEN.

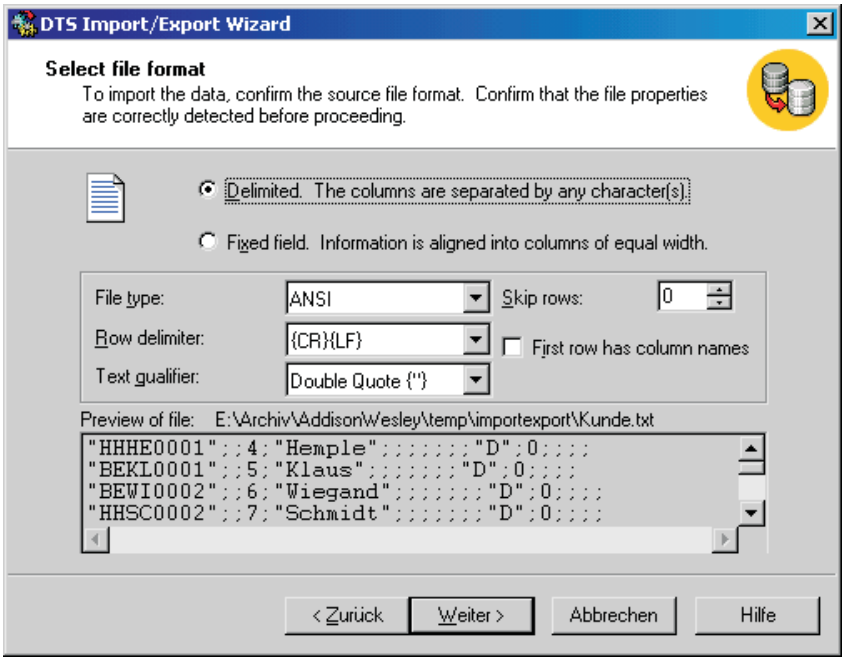


Abbildung 8.17: Datenimport im SQL Server

DTS stellt weitere Funktionalitäten, wie das Erzeugen von Metadaten, für die Weiterverarbeitung bei Datawarehouses oder Datamarts zur Verfügung. Außerdem gibt es mit dem DTS-Abfrage-Designer ein Werkzeug mit grafischer Oberfläche, mit dem der Anfänger sich leicht SQL-Abfragen erstellen kann.

8.5.5 Datenexport

Für den Datenexport sollte zunächst die Frage nach den angebotenen Exportformaten im Vordergrund stehen. Gibt es das Format, das auch vom Zielsystem gelesen werden kann, so wird sich der Datenexport als leichtes Spiel erweisen. Auch für den Export soll in einem Beispiel demonstriert werden, wie beim SQL Server mit dem Data Transformation Service gearbeitet wird. Adressdaten aus der Beispieldatenbank sollen in eine *Excel-Tabelle* exportiert werden. Vor dem Export müssen Sie die *Excel-Datei* in einem Export-Verzeichnis erstellen. Legen Sie einfach eine leere *Excel-Datei* mit dem Namen ADRESSEXPORT.XLS in das ausgewählte Verzeichnis.

1. Starten Sie DTS über START/PROGRAMME/MICROSOFT SQL SERVER/DATEN IMPORTIEREN UND EXPORTIEREN.
2. Klicken Sie im ersten Dialogfeld auf die Schaltfläche WEITER.
3. Geben Sie für die *Datenquelle* den Server und die Beispieldatenbank an (siehe Abbildung 8.18).

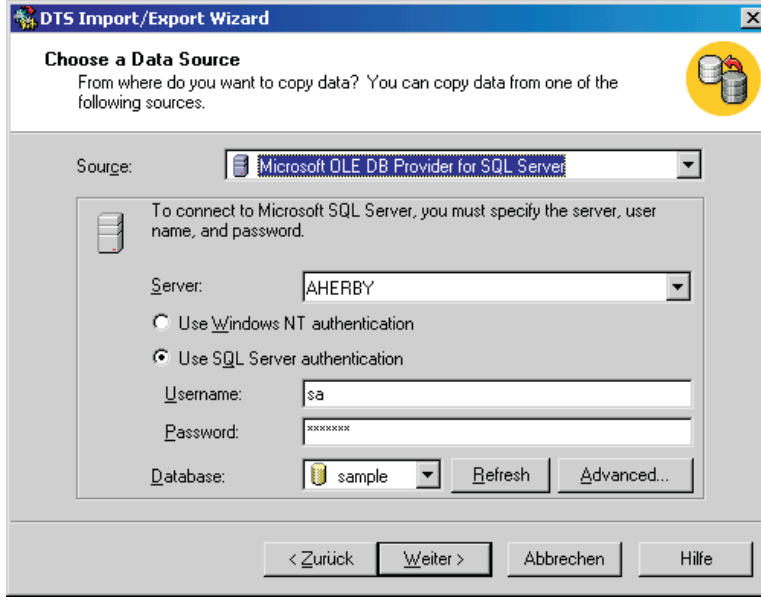


Abbildung 8.18: Datenexport mit dem SQL-Server

4. Geben Sie für das *Zielsystem* MICROSOFT EXCEL an.
5. Wählen Sie den *Dateinamen* und das *Verzeichnis* für die *Exportdatei.*

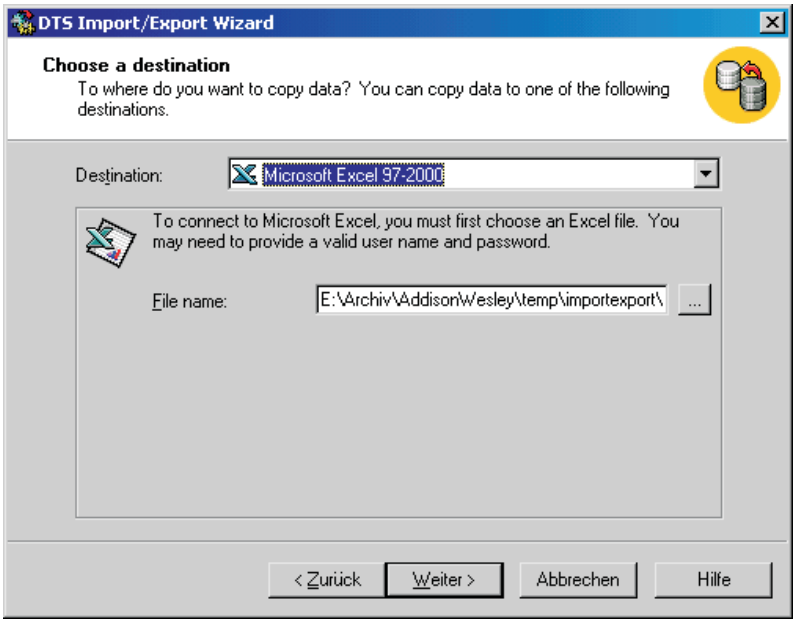


Abbildung 8.19: Angabe des Exportformates in DTS

6. Selektieren Sie im nächsten Dialogfeld die Option KOPIEREN AUS TABELLEN UND SICHTEN.
7. Markieren Sie die *Quelltabellen*.
8. Geben Sie den *Zeitpunkt* für den Export an oder exportieren Sie direkt nach Fertigstellung des Importauftrages.
9. Beenden Sie den Vorgang durch Aktivieren der Schaltfläche FERTIG STELLEN.

Sie können die Daten unmittelbar nach der Ausführung als Excel-Tabelle einsehen. Öffnen Sie Microsoft Excel und laden Sie die Datei ADRESSEXPORT.XLS. Sie werden dann eine Tabelle vorliegen haben, wie sie in dem Bild 8.20 gezeigt wird. Wahrscheinlich werden Sie mit dem Layout der Tabelle nicht ganz zufrieden sein. Es steht Ihnen offen, dies nach Ihren Wünschen anzupassen.

adressexport

	A	D	E	G	I	J	K	L
1	AdressNr	Name	Kurzname	StrassePostfach	Land	Plz	Ort	AnschriftAnrede
2	4	Heiner	Heiner	Meisenweg 3	D	40032	Essen	Herr
3	5	Klaus	Klaus	Wacholderweg 23	D	34550	Herborn	Herr
4	6	Förster	Förster	Ludwigstr. 234	D	20001	Hamburg	Frau
5								
6								
7								

Tabelle1 Tabelle2 Tabelle3 adresse

Abbildung 8.20: Exportdaten in Excel geöffnet

8.6 Synchronisieren der Daten durch die Replikation

In der Einführung wurde ein Szenario entworfen, in dem ein Unternehmen mehrere Niederlassungen hat, die jeweils mit einem Datenbankserver bestückt sein sollen. Nun gibt es Daten, die sind nur für die jeweiligen Niederlassungen relevant. Man denke dabei an Adressdaten der Kunden oder Mitarbeiterdaten. Um Kosten zu senken, wird die Buchhaltung oft zentralisiert durchgeführt. Dafür ist es notwendig, dass Daten aus den verschiedenen Niederlassungen regelmäßig abgeglichen werden. Eine Möglichkeit, diesen Abgleich zu realisieren, bietet die *Datenreplikation*.

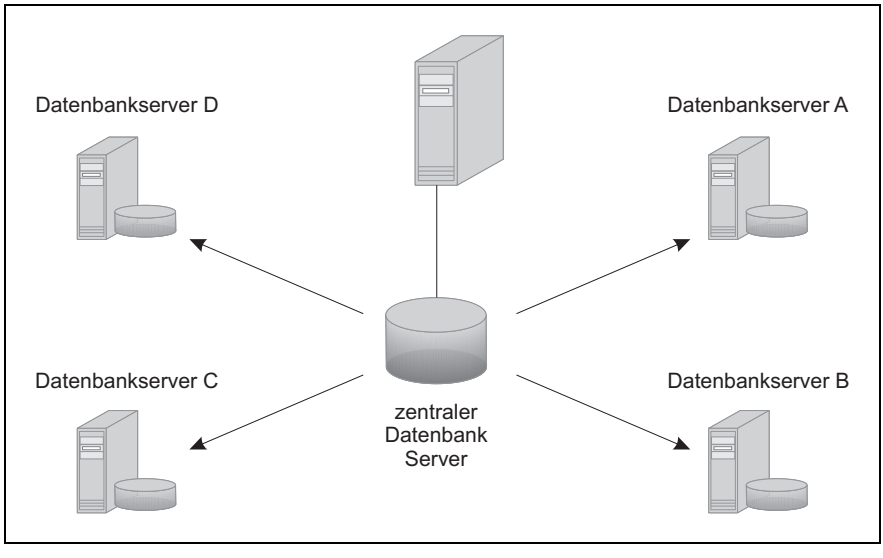


Abbildung 8.21: Übersicht zur Datenreplikation

8.6.1 Übersicht zur Replikation

Mithilfe der Replikation können Daten an verschiedene Standorte über Netzwerk verteilt werden. Es handelt sich um eine Technik zum Kopieren und Verteilen von Daten und Datenbankobjekten aus einer Datenbank in eine andere. Datenbanken werden synchronisiert, um die Konsistenz der Daten sicherzustellen. Datenreplikation bringt beim Arbeiten mit mehreren Datenbanken einige Vorteile:

- Hohe Verfügbarkeit an Daten,
- Unterstützung von Data Warehousing,
- dezentrales Skalieren und Durchsuchen der Daten,
- Ausfallsicherheit durch das Kopieren der Daten,
- größere Eigenständigkeit der einzelnen Standorte,
- Kapazitätseinsparungen.

Es gibt zwei fundamentale Kategorien von Replikationsvorgängen. *Synchrone* und *asynchrone Replikation*. Bei der synchronen Replikation werden unmittelbar nach einer Veränderung an den Quelldaten die replizierten Daten abgeglichen. Nachdem ein Datenbankserver Änderungen an den Datenbeständen vorgenommen hat, wird eine Nachricht an den zentralen Server gesendet. Dadurch wird ein hohes Maß an *Integrität* sichergestellt, denn jede Transaktion wird zentral gesteuert und überprüft. Gleichzeitig benötigt die synchrone Replikation einen hohen Aufwand an Verwaltungsaktionen, die zu Lasten der Performance und der Verfügbarkeit gehen. Die synchrone Replikation wird immer dann angewendet, wenn Daten sofort synchronisiert werden müssen (z.B. Finanzdaten oder Wetterdaten).

Die asynchrone Replikation unterscheidet sich dahingehend, dass die Daten zunächst in der Quelldatenbank geändert werden und zu einem späteren Zeitpunkt, die Daten in der oder in den Zieldatenbanken aktualisiert werden. In Abhängigkeit von der vorgenommenen Konfiguration kann die Zeitspanne zwischen der Veränderung an der Quelldatenbank und dem Replikationsvorgang einige Sekunden, Stunden oder Tage betragen. Ebenso ist es von der Konfiguration abhängig, ob gleichzeitig alle Niederlassungen an der Replikation beteiligt sind oder nur einzelne. Ist eine Niederlassung vorübergehend nicht erreichbar, kann dies protokolliert werden und ein erneuter Replikationvorgang kann zu einem späteren Zeitpunkt erneut gestartet werden. Der Vorteil gegenüber der synchronen Replikation liegt besonders in der besseren *Abfragezeit* und deutlich weniger Bedarf an Kapazitäten für die Datenübertragung. Dafür muss bei der asynchronen Replikation ein Plan erstellt werden, mit welchen Maßnahmen die Datenintegrität gewährleistet werden kann.

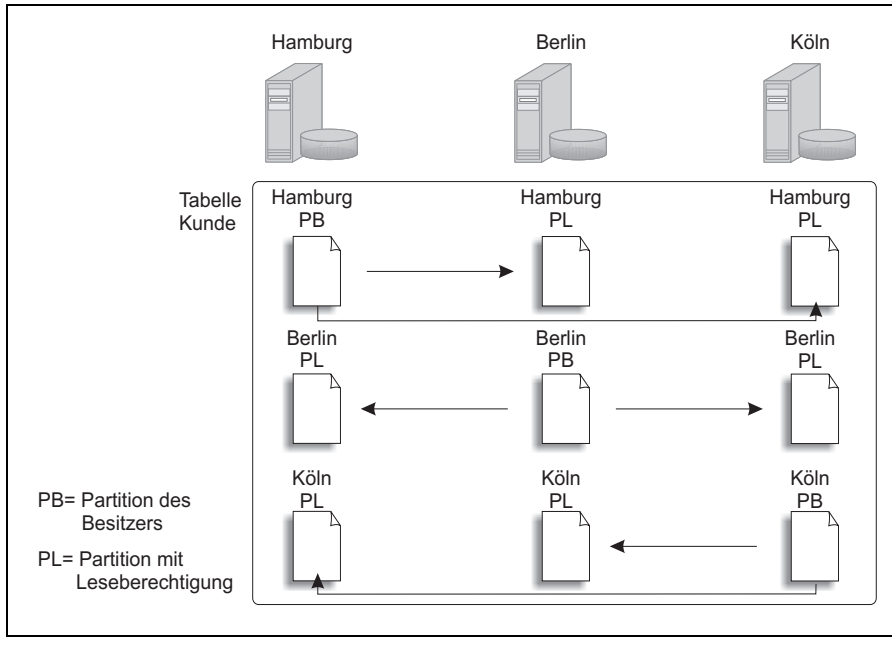


Abbildung 8.22: Replikationsmodell für die Beispieldatenbank

8.6.2 Replikationsbeispiel

An dieser Stelle wollen wir uns wieder der Umsetzung in die Praxis zuwenden. Die Daten aus der Beispieldatenbank sollen in allen Niederlassungen repliziert werden. Für die Replikation nehmen wir an, es handelt sich um Datenbanken verschiedener Hersteller, d.h. wir haben heterogene Datenquellen. In der Zentrale in Berlin sollen die Daten mit dem SQL Server verwaltet werden, im Hamburg mit PostgreSQL und in Köln werden die Daten in Access gepflegt. Es soll ausreichen, die Datenbanken einmal täglich nach Feierabend zu replizieren, d.h. es wird mit eine asynchronen Replikation gearbeitet. Der SQL Server in Berlin ist das zentrale Element, deshalb wird mit dem *Replikationsmodell* von Microsoft gearbeitet.

Replikationstopologie beim MS SQL Server

In Anlehnung an die *Topologie* aus dem Verlagswesen, wird mit den Komponenten gearbeitet, die wir in Abbildung 8.23 sehen. Der *Verleger* ist ein Datenbankserver, der andere Datenbanken mit Daten versorgt. Diese Daten werden auch *Publikation* genannt. Der Verleger kann nun eine *Publikation* oder mehrere dem *Verteiler* zur Verfügung stellen, bei dem es sich wiederum um einen Datenbankserver handelt. Der Verteiler kann mit dem Verleger identisch sein, ein Datenbankserver nimmt dann sowohl die Aufgaben des Verlegers wahr und kontrolliert zusätzlich die Verlaufsdaten der

Transaktionen und die Metadaten. In diesem Fall spricht man von einem *lokalen Verteiler*. Sind die Aufgaben getrennt, so spricht man von einem *Remoteverteiler*.

Die *Abonnenten* sind schließlich die Empfänger der Publikationen. Sie erhalten nur die Publikationen, die sie abonniert haben, d.h. die Abonnement festgehaltenen Kriterien entscheiden wann und welche Artikel dem Abonnenten übermittelt werden. Je nach Replikationsmodell ist es auch möglich, dass die Abonnenten Änderungen an den Verleger übertragen. Es gibt zwei verschiedenen Arten von Abonnements. Werden die Publikationen vom Verleger zugewiesen, dann nennt man dies ein *Push-Abonnement*. Wird die Publikation dagegen von den Abonnenten angefordert, so spricht man auch von einem *Pull-Abonnement*. Auch Mischformen aus beiden Typen sind möglich.

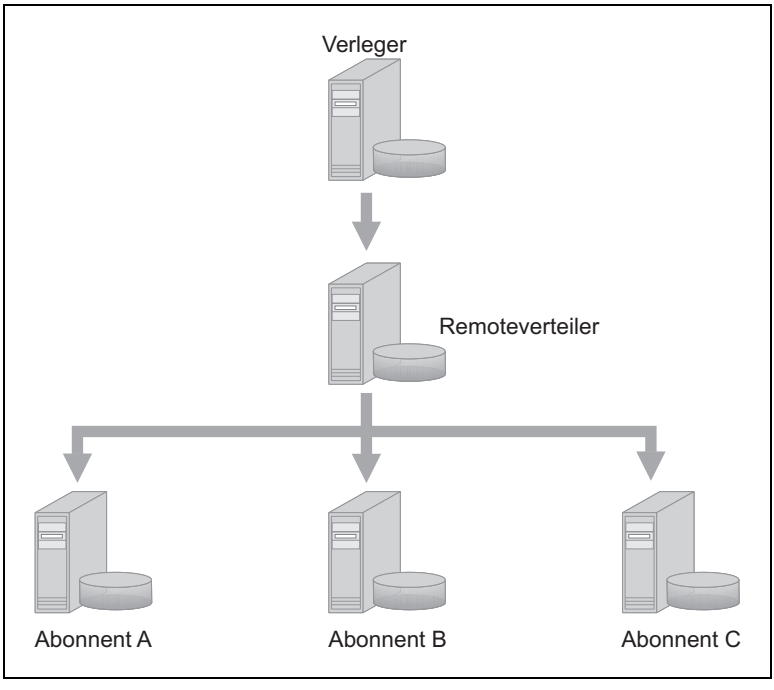


Abbildung 8.23: Begriffsklärung bei der MS Replikationstopologie

Replikationstypen beim MS SQL Server

Der SQL Server kennt drei Replikationstypen: *Snapshotreplikation*, *Transaktionsreplikation* und *Mergereplikation*. Je nach Typus kann eine synchrone oder asynchrone Replikation durchgeführt werden.

Snapshotreplikation

Bei der *Snapshotreplikation* wird von Daten und Datenbankobjekte ein »Schnappschuss« gemacht und die Daten werden genau in dem Status kopiert und verteilt, der zum Zeitpunkt des »Schnappschusses« bestand. Änderungen werden nicht inkrementell an die Abonennten weitergeleitet, deshalb müssen Änderungen auch nicht überwacht werden. Abonnenten werden mit einer vollständigen Aktualisierung des Datensatzes aktualisiert, nicht mit einzelnen Transaktionen. Der Replikationsvorgang kann mitunter etwas länger dauern, weil jeweils eine komplette Publikation an die Abonnenten übergeben wird. Dieser Replikationstyp wird in der Praxis nicht so häufig verwendet wie die beiden folgenden.

Transaktionsreplikation

Auch bei der *Transaktionsreplikation* wird zunächst ein Anfangssnapshot der Daten an Abonnenten weitergeleitet. Zusätzlich werden danach Datenänderungen, die auf dem Verleger vorgenommen werden, aufgezeichnet und die einzelnen Transaktionen an Abonnenten weitergeleitet.

INSERT-, UPDATE- und DELETE-Anweisungen sowie Änderungen an Ausführungen von gespeicherten Prozeduren und indizierten Sichten werden vom DBMS überwacht. Transaktionen die sich auf replizierte Objekte auswirken werden gespeichert und können direkt oder nach einem definierten Zeitintervall an die Abonnenten übertragen werden. Dieser Replikationstyp ermöglicht die synchrone Replikation. Abonnenten und Verleger können nahezu zeitgleich dieselben Werte haben.

Mergereplikation

Die *Mergereplikation* ermöglicht den verschiedenen Standorten, eigenständig (online oder offline) zu arbeiten und an mehreren Standorten vorgenommene Datenänderungen zu einem späteren Zeitpunkt in einem einzelnen einheitlichen Ergebnis zusammenzuführen. Der Anfangssnapshot wird auf Abonnenten angewendet. SQL Server verfolgt dann Änderungen an publizierten Daten auf dem Verleger und auf den Abonnenten. Die Daten werden zwischen Servern entweder zum geplanten Zeitpunkt oder bei Bedarf synchronisiert. Aktualisierungen werden unabhängig (ohne Commitprotokoll) auf mehr als einem Server vorgenommen; somit wurden dieselben Daten möglicherweise vom Verleger oder von mehr als einem Abonnenten aktualisiert. Hierbei muss beachtet werden, dass beim Zusammenführen der Datenänderungen Konflikte auftreten können. Besondere Planungen, welche die Integrität bewahren, sind erforderlich.

Das DBMS bietet einige Standardoptionen und benutzerdefinierte Optionen für die Konfliktlösung, die definiert werden können, wenn eine Mergepublikation konfiguriert wird. Tritt ein Konflikt auf, wird ein *Konfliktlöser* vom *Merge-Agent* aufgerufen, der bestimmt, welche Daten akzeptiert und an andere Standorte weitergegeben werden.

Erstellen eines Replikationsmodells

Aus den gegebenen Anforderungen muss ein Modell für die Replikation gewählt werden. Da die Daten täglich abgeglichen werden sollen, eignet sich das Modell der Mergereplikation am besten für den Zweck. Damit der Aufbau nicht unnötig kompliziert wird, soll auch von einem lokalen Verteiler ausgegangen werden. Verleger- und Verteileraufgaben werden also vom DBMS in Berlin ausgeführt. Die Zusammenhänge sind in Bild 8.24 dargestellt.

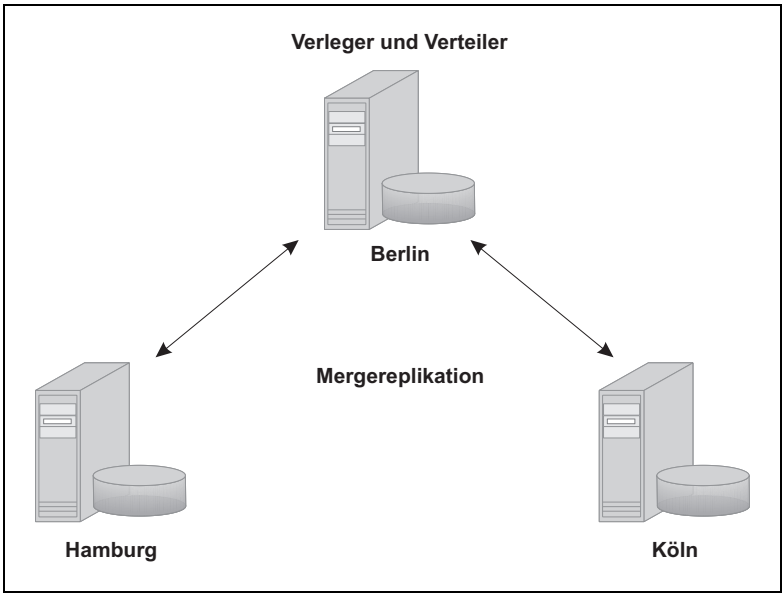


Abbildung 8.24: Replikationsmodell der Beispieldatenbank für den SQL Server

Implementieren der Replikation

Das Replikationsmodell ist erstellt, somit können die weiteren Schritte zur Implementierung der Replikation angegangen werden. Die Vorgehensweise kann in logische Arbeitsschritte unterteilt werden:

Konfiguration

1. Verbindung zu Datenquellen sicherstellen (siehe Kapitel 7).
2. Konsolenstruktur im *Enterprise Manager* erweitern bis REPLIKATION erscheint.
3. Mit der rechten Maustaste auf REPLIKATION klicken.
4. In Kontextmenü PUBLIZIERUNG, ABONNENTEN UND VERTEILUNG KONFIGURIEREN auswählen.

5. Im ersten Fenster des Assistenten auf WEITER klicken.
6. Es muss ein Server als Verteiler eingestellt werden (hier ist der Verleger gleich dem Verteiler).
7. Für die weitere Verteilerkonfiguration werden die Standardeinstellungen verwendet.
8. Klicken Sie im nächsten Fenster auf FERTIGSTELLEN.
9. Nachdem Sie den Verteiler konfiguriert haben, müssen Sie Abonnenten definieren. Klicken Sie dazu nochmals mit der rechten Maustaste in der Konsole auf REPLIKATION.
10. In Kontextmenü PUBLIZIERUNG, ABONNENTEN UND VERTEILUNG KONFIGURIEREN auswählen.
11. Nehmen Sie in der Registerkarte PUBLIKATIONSDATENBANK die Einstellungen wie in Bild 8.25 vor.

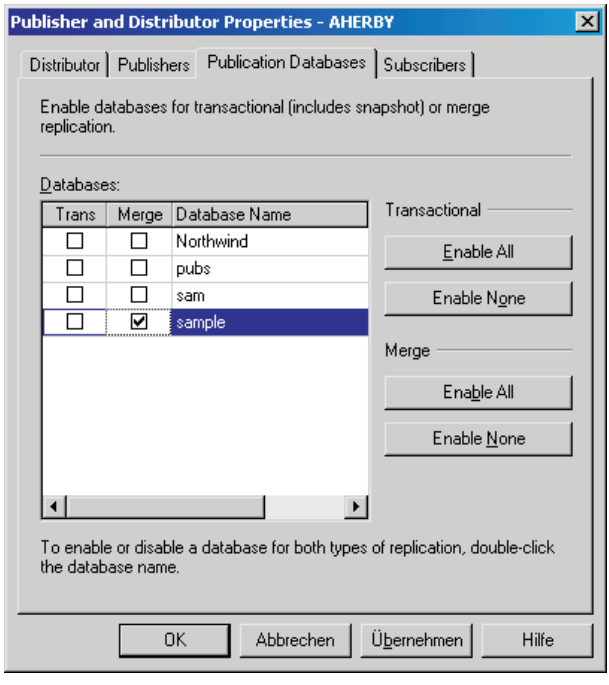


Abbildung 8.25: Einstellungen für die Publikationsdatenbank

12. Klicken Sie in der Registerkarte ABONNENTEN (Subscribers) auf die Schaltfläche NEU.

13. Im nächsten Dialogfeld geben Sie die Verbindungsinformation ein (siehe Abbildung 8.26).

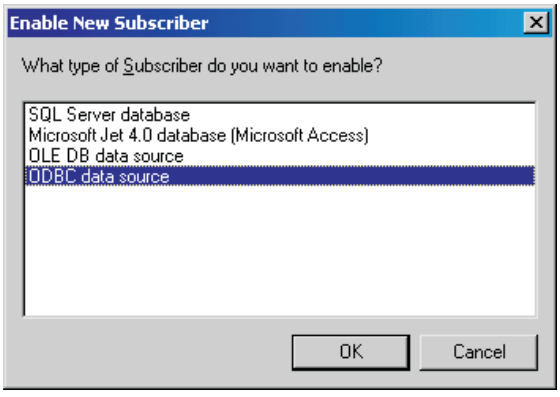


Abbildung 8.26: Anlegen eines neuen Abonnenten

14. Wählen Sie aus der Auswahlliste die Datenquelle, die als Abonnent zur Verfügung steht (siehe Abbildung 8.27).

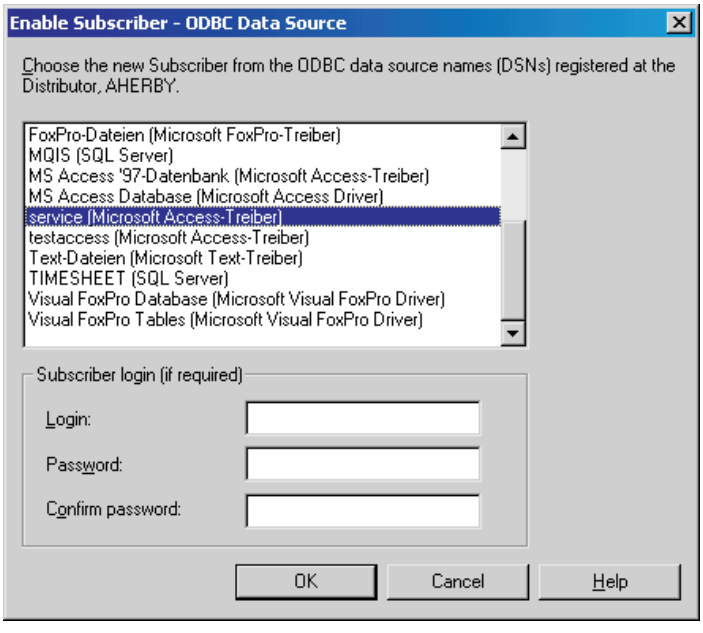


Abbildung 8.27: Auswahl der Datenquelle

15. Sie haben nun einen weiteren Abonnenten hinzugefügt. Die gleiche Vorgehensweise gilt für das Hinzufügen weiterer Abonnenten.

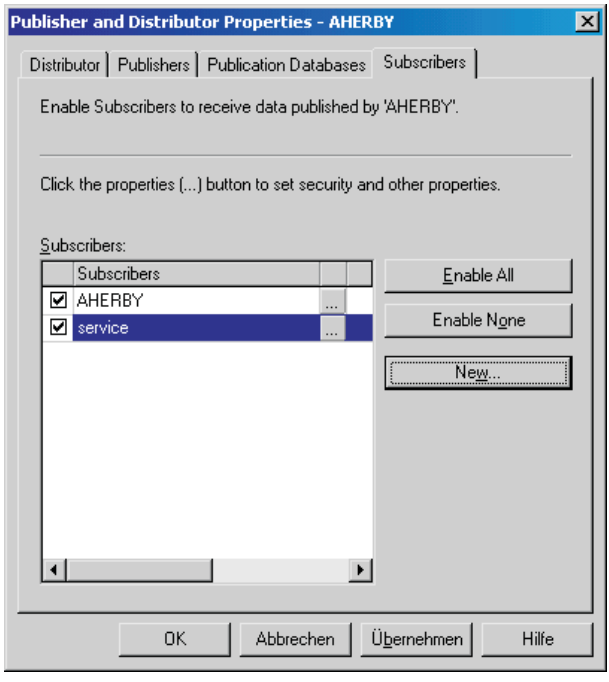


Abbildung 8.28: Register Abonnenten

Erzeugen und Anwenden des Anfangssnapshots

1. Erstellen Sie eine neue Publikation, indem Sie in der Konsole mit der rechten Maustaste auf PUBLIKATIONEN und im Kontextmenü auf NEUE PUBLIKATION klicken.
2. Klicken Sie im ersten Dialogfeld des Assistenten auf WEITER.
3. Selektieren Sie die *Publikationsdatenbank* aus der Auswahlliste gemäß dem Beispiel in Abbildung 8.29.
4. Wurde bereits eine Publikation erstellt, können Sie diese als *Muster* für eine neue Publikation verwenden oder Sie definieren neue Artikel für die Publikation.
5. Entscheiden Sie sich für den Replikationstypen (siehe Abbildung 8.30).

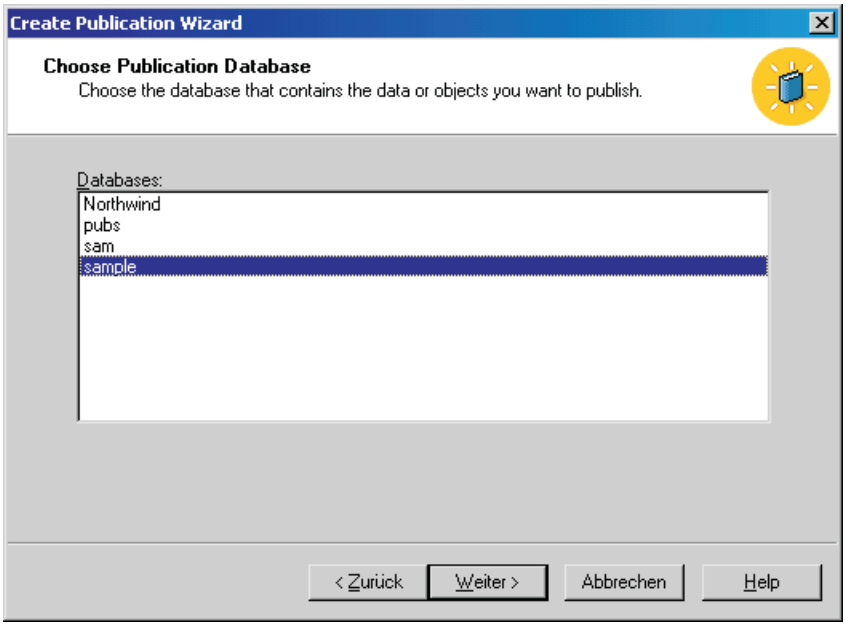


Abbildung 8.29: Auswahl der Datenbank im Assistenten für Publikationen

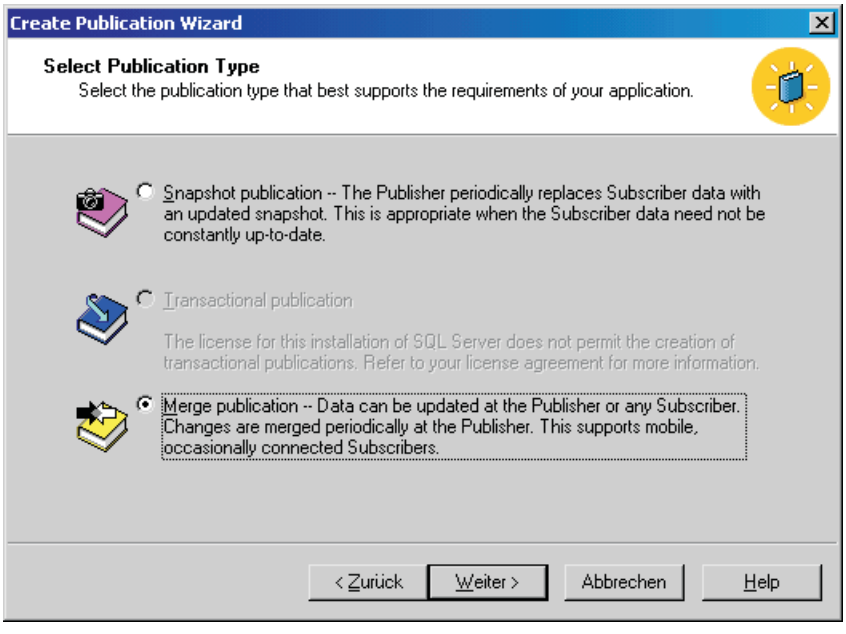


Abbildung 8.30: Auswahl eines Replikationstyps

6. Wählen Sie die Typen, die als Abonnenten möglich sind. Für unser Replikationsbeispiel können Sie die Einstellungen, wie in Bild 8.31 gezeigt, vornehmen.

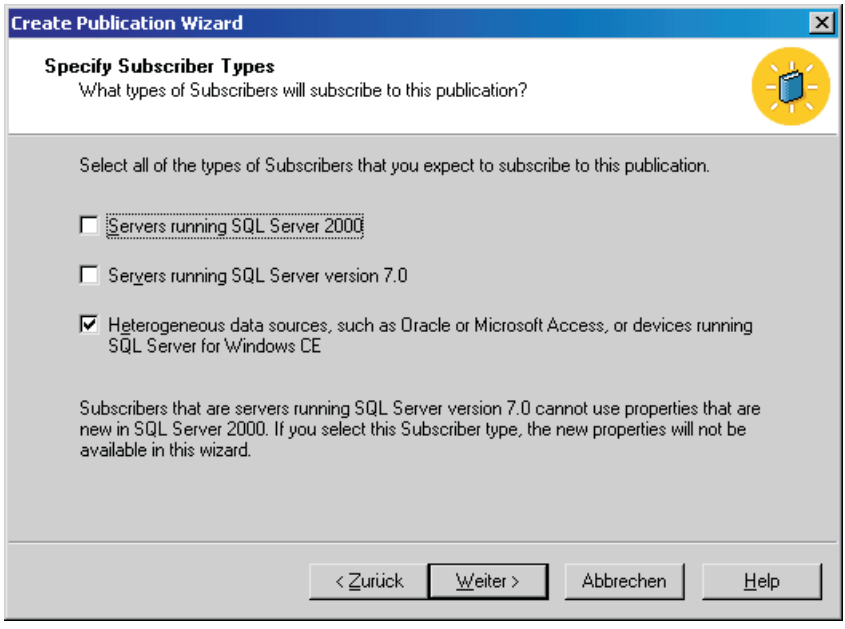


Abbildung 8.31: Typen für die Abonnenten

7. Geben Sie die *Artikel* (Tabellen, gespeicherte Prozeduren und Sichten) an, die publiziert werden sollen (siehe Abbildung 8.32).
8. Geben Sie im nächsten Dialogfeld den *Namen* und eine *Beschreibung* für die Publikation an (siehe Abbildung 8.33).
9. Definieren Sie einen *Datenfilter* oder wählen Sie die Standardeinstellung.
10. Klicken Sie auf FERTIGSTELLEN, um die Publikation vom DBMS erzeugen zu lassen.
11. Die Publikation wird erstellt und in der Konsolenstruktur finden Sie einen Eintrag für die erzeugte Publikation. Sie können weitere Publikationen anlegen und vorhandene ändern.

Für die erzeugte Publikation ein Pushabonnement erstellen

1. Klicken Sie in der Konsolenstruktur auf das Symbol für die soeben erstellte Publikation mit der rechten Maustaste.
2. Im Kontextmenü klicken Sie auf PUSH FÜR NEUES ABONNEMENT ERSTELLEN.

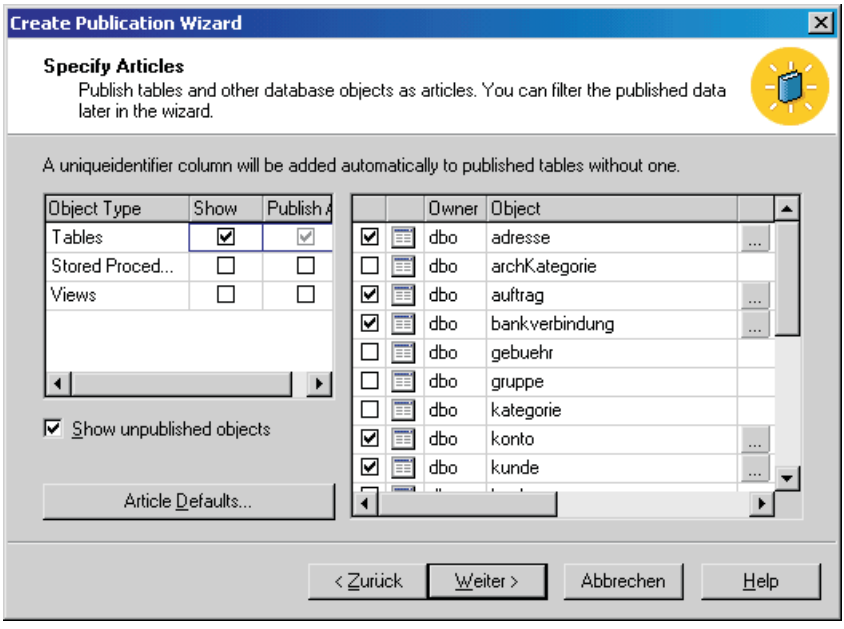


Abbildung 8.32: Auswahl der zu publizierenden Artikel

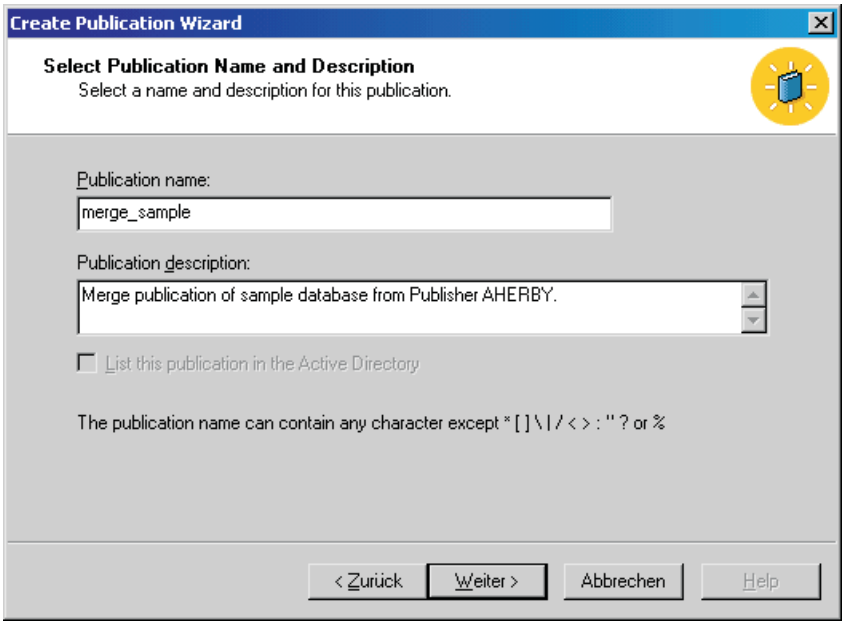


Abbildung 8.33: Beschreibungsangaben für die Publikation

3. Im ersten Dialogfeld des Assistenten klicken Sie auf WEITER.
4. Wählen Sie die *Abonnenten* aus der vorgegebenen Liste.
5. Geben Sie einen *Namen* für die Abonnementdatenbank an (für heterogene Datenquellen ist ein bereits erstellter Anfangssnapshot erforderlich).

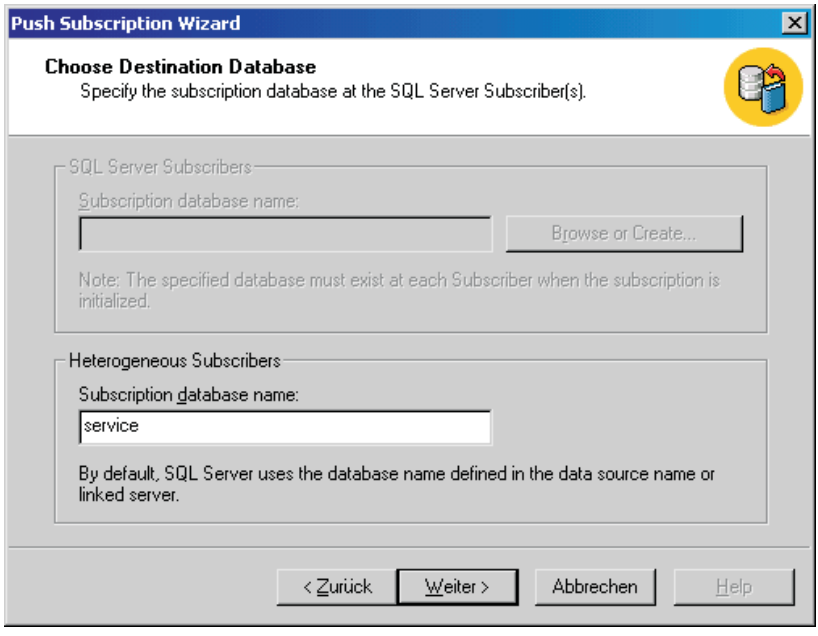


Abbildung 8.34: Namensangabe für die Abonenntendatenbank

6. Nehmen Sie die *Initialisierung* vor.
7. Erstellen Sie ggf. einen *Terminplan* (siehe Abbildung 8.35).
8. Wählen Sie die notwendigen *Dienste* und überprüfen Sie ihren Status (siehe Abbildung 8.36).
9. Klicken Sie auf FERTIGSTELLEN und der Abonnent wird mit der Publikation verbunden.

Mit dem *Snapshot-Agenten* kann das Anfangssnapshot erstellt werden. Dieser kann manuell oder über einen Terminplan aktiviert werden. Das Anfangssnapshot wird angewendet, indem der Snapshot-Agent Schema und Daten von der Verlegerdatenbank auf die Abonnentendatenbank kopiert. Die Replikation von geänderten Daten wird erst dann durchgeführt, wenn die Mergereplikation sichergestellt hat, dass der Abonnent über den aktuellsten Snapshot des Tabellenschemas und den Tabellendaten verfügt, die generiert wurden.

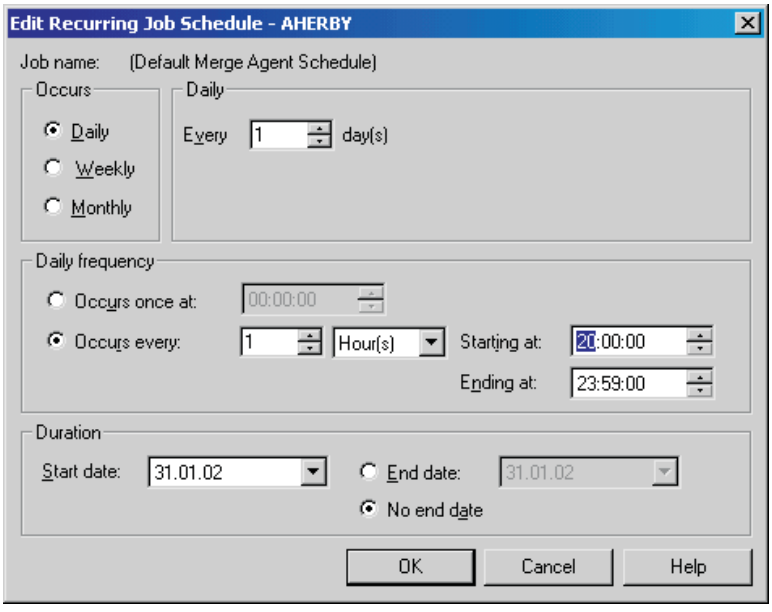


Abbildung 8.35: Erstellen eines Terminplans

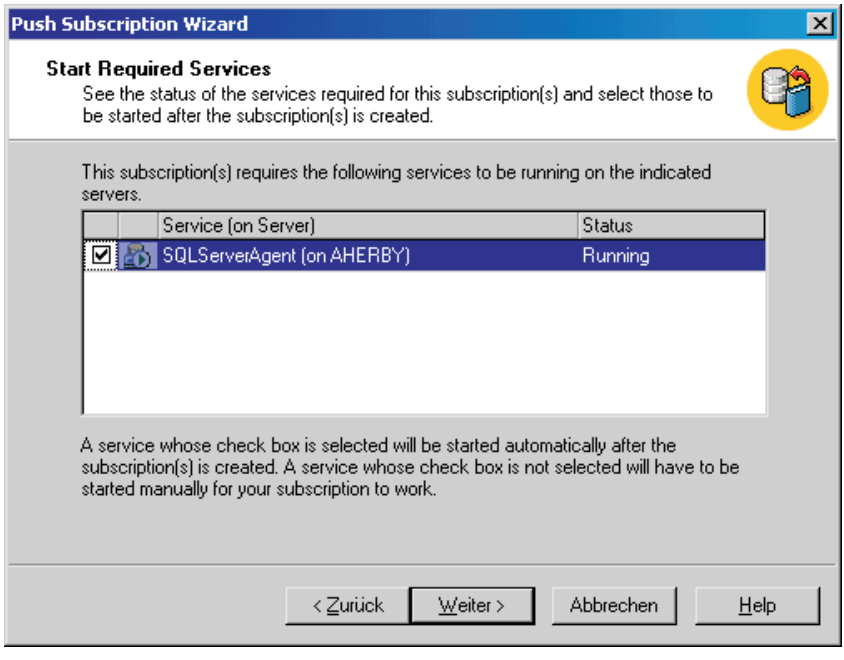


Abbildung 8.36: Angabe der verwendeten Dienste

Änderung an replizierten Daten

Wurde der Anfangssnapshot definiert und angewandt, so überprüfen Systemtrigger die Datenbankobjekte, welche an der Replikation beteiligt sind. Bei INSERT-, UPDATE- und DELETE-Anweisungen werden Datensätze in Systemtabellen geschrieben. Wenn die Verbindung zwischen Verleger und Abonnent wieder hergestellt und der Merge-Agent ausgeführt wird, sammelt der Merge-Agent alle nicht abgeschlossenen Zeilenänderungen (mit neuen Generierungswerten) in mindestens einer Gruppe und weist Generierungswerte zu, die höher als alle vorherigen Generierungen sind. Dadurch kann der Merge-Agent Änderungen an verschiedenen Tabellen in Batches (Stapel) sammeln und diese verarbeiten, um eine höhere Effizienz über langsame Netzwerke zu erreichen.

Bei einer Remoteverbindung müssen Sie eine DCOM-Konfiguration (Distributed Components Object Model) über Ihr Betriebssystem ausführen, bevor Sie den Merge-Agenten einsetzten können.

Synchronisieren und Weitergeben von Datenänderungen

Bei der Synchronisierung werden geänderte Datenwerte auf andere Standorte repliziert und mit Änderungen, die an anderen Standorten vorgenommen wurden, zusammengeführt. Synchronisierungen können im Abstand von Minuten, Tagen oder sogar Wochen ausgeführt werden und werden im Zeitplan des Merge-Agents definiert. Werden die Abonnements nicht innerhalb der Aufbewahrungsdauer zwischen Verleger und Abonnenten synchronisiert, so gelten sie als abgelaufen und müssen neu initialisiert werden.

Eine Synchronisation kann manuell über den Enterprise Manager gestartet werden:

1. In der Konsolenstruktur den Ordner PUBLIKATION öffnen.
2. Auf die gewünschte Replikation im rechten Fenster mit der rechten Maustaste klicken.
3. Klicken Sie dann im Kontextmenü auf SYNCHRONISIERUNG STARTEN.

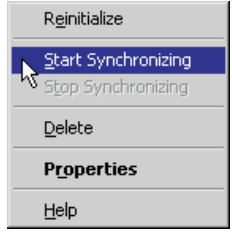


Abbildung 8.37: Start der Synchronisierung

Durch die Einstellung SYNCHRONISIERUNG OPTIMIEREN im Publikationserstellungs-Assistenten werden mehr Daten in der Verlegerdatenbank gespeichert und somit der Netzwerkverkehr minimiert. Die meisten der hier beschriebenen Vorgänge, die exemplarisch im Enterprise Manager durchgeführt wurden, lassen sich auch mittels Systemprozeduren und TSQL erledigen.

8.6.3 Verbesserung der Replikationsleistung

Die allgemeine Leistung kann für alle Replikationsarten in Ihrer Anwendung und im Netzwerk auf die folgende Weise verbessert werden:

Hinzufügen von Arbeitsspeicher für die Server, die für die Replikation verwendet werden.

Die Replikationsleistung kann optimiert werden, indem Sie Arbeitsspeichererweiterungen für die Server durchführen, die für die Replikation verwendet werden.

Festlegen einer Mindestmenge an Arbeitsspeicher, der für das DBMS reserviert wird.

Standardmäßig ändert das DBMS seinen Arbeitsspeicherbedarf auf Basis der verfügbaren Systemressourcen dynamisch. Wenn verhindert werden soll, dass während Replikationsaktivitäten nur wenig Arbeitsspeicher zur Verfügung steht, kann über Datenbankoptionen ein Minimum an Arbeitsspeicher festgelegt werden. Falls der Server ein Remoteverteiler oder ein kombinierter Verleger und Verteiler ist, müssen Sie mindestens 16 Megabyte Arbeitsspeicher zuweisen.

Verwenden eines separaten Laufwerkes für das Transaktionsprotokoll für alle an der Replikation beteiligten Datenbanken.

Dies betrifft die Publikations-, die Verteilungs- und die Abonnementdatenbank. Sie können die Zeit verringern, die zum Schreiben von Transaktionen benötigt wird, indem Sie die Protokolldateien auf ein anderes Laufwerk schreiben als das, das zum Speichern der Datenbank verwendet wird.

Computer mit mehreren Prozessoren verwenden.

In manchen Fälle, z.B. bei hoher CPU-Nutzung, könnten schnellere oder mehrere CPUs (symmetrisches Multiprocessing) eine Lösung sein.

Nur die erforderlichen Datenmengen publizieren.

Es besteht die Gefahr, aus reiner Lust am Replizieren, die Datenmenge auf Bereiche auszudehnen, bei denen eine Replikation gar nicht erforderlich ist. Dies kann weitere Ressourcen innerhalb der Verteilungsdatenbanken und Snapshotdateien beanspruchen und den Durchsatz für erforderliche Dateien verringern.

Staffeln der Verteilungsausführung.

Ein Verteiler kann Transaktionen an eine größere Anzahl von Abonnenten verteilen. Wird für die Replikation ein Zeitplan verwendet, so können verschiedene Abonnenten zeitlich gestaffelt bedient werden.

Trigger und Indizes beim Abonnenten.

Benutzerdefinierte Trigger und Indizes auf Seiten des Abonnenten können den Datendurchsatz nachteilig beeinflussen.

Schnelles Netzwerk.

Die Weitergabe von Änderungen an den Abonnenten kann deutlich beschleunigt werden, wenn ein schnelles Netzwerk mit 100 Mbit/s oder mehr verwendet wird.

9 Sicherheit und Wartung

Besonders wichtig bei der Umsetzung des Softwareprojektes ist die Berücksichtigung der Datensicherheit. Wie schütze ich meine Daten vor unbefugtem Eingriff? Welche Sicherheitsprobleme kann es im Internet geben? Dieses Kapitel gibt Ihnen nicht nur einen Überblick über die Sicherheitsmechanismen der Datenbanktechnologie, sondern stellt auch allgemeine Sicherheitskonzepte vor, die in einem Unternehmen umgesetzt werden sollten. Daten können auf verschiedenen Ebenen gesperrt werden. Lernen Sie, wie Sie für die Mitarbeiter Ihres Unternehmens Benutzerkonten einrichten, damit diese sich bei der Datenbankanwendung anmelden können. Backup- und Recovery-Vorgänge werden zur Vermeidung von ungewollten Datenverlusten beschrieben. Es gibt einige Routineaufgaben, die der Datenbankadministrator manuell oder automatisch an der produktiven Datenbank durchführen muss. Machen Sie sich das Leben leichter, indem Sie Wartungspläne für diese Tätigkeiten erstellen.

9.1 Murphys Gesetz – der Supergau

»Alles, was schief gehen kann, geht auch schief.«
E. A. Murphy

US Air Force Captain *Edward A. Murphy*, Konstrukteur am Wright Field Aircraft Lab, wurde 1949 damit beauftragt, eine Ausrüstung für Piloten zu entwickeln. Mit dieser Ausrüstung, die von einem Testpiloten getragen wurde, sollten Messungen durchgeführt werden, die Aufschluss darüber geben sollten, wie viel Beschleunigung der menschliche Körper aushalten kann. Der Messwandler war mit 16 Sensoren versehen, die die Belastung messen sollten. Nachdem die Entwicklung abgeschlossen war, wurde der Messwandler zum Einsatzort geschickt, wo das Gerät an einem Raketenschlitten angebracht wurde.

Murphy hörte schließlich, dass der Test schief gegangen war. Da man seinen Messwandler dafür verantwortlich machte, und es eine sehr teure Panne war, begab er sich persönlich an den Einsatzort. Er vermutete sogleich, dass die Störungsquelle der Anschluss des Dehnungsmessers sei. »Ein Dehnungsmesser lässt sich nur auf zwei Arten anschließen: auf die richtige Art oder in 90 Grad Abweichung von der richtigen Art.«

Bei seinen Nachforschungen stellte Murphy fest, dass der Dehnungsmesser tatsächlich falsch angeschlossen worden war. So formulierte er die ursprüngliche Form des Murphy-Gesetzes: «*Wenn es zwei oder mehr Möglichkeiten gibt, etwas zu tun, und wenn eine dieser Möglichkeiten zu einer Katastrophe führt, dann wird sich irgendjemand für genau diese Möglichkeit entscheiden.*»

Seitdem wurde diese Aussage in viele Lebensbereiche adaptiert und immer wieder wurde die Richtigkeit der Kernaussage festgestellt. Für die Thematik des Buches hat der Autor, mit einem Hauch Ironie, drei Kernaussagen formuliert:

1. Der Anwender wird zielsicher den Weg finden, der es ihm erlaubt, alle relevanten Daten zu löschen.
2. Die geschützten Daten sind immer die interessantesten.
3. Es gibt immer eine Sicherheitslücke, die Datenverlust oder Datenmanipulation ermöglicht.

Dies ist natürlich eine übertrieben düstere Sicht der Dinge. Außerdem sind nicht alle Daten so brisant, dass sie ein Maximum an Sicherheitsvorkehrungen benötigen. Trotzdem ist es in vielerlei Hinsicht lohnend, sich über ausreichende Sicherheitsmaßnahmen Gedanken zu machen. In einigen Bereichen ist es sogar ein absolutes Muss, dass man sich an geltende Sicherheitsrichtlinien hält.

9.2 Sicherheitsarchitektur

Datensicherung heißt, technische und organisatorische Mittel zur Verfügung zu stellen, um die Sicherheit der Informationstechnik und die Datenbestände und Datenverarbeitungsabläufe in jeder Phase zu gewährleisten. Dabei ist menschliches Handeln laut statistischen Erhebungen die größte Gefahrenquelle. Hardware- und Softwareversagen bilden weiteres Gefährdungspotenzial.

9.2.1 Schutz vor unbefugter Datennutzung

Beim Schutz der Daten vor unbefugter Nutzung müssen die Daten nach Schutzbedürftigkeit bewertet werden, denn nicht alle Daten benötigen das höchste Maß an Schutz. In Abbildung 9.1 ist ein *Schutzstufenmodell* dargestellt. Je höher die Schutzstufe, desto mehr sollte für die Sicherung der Daten aufgewendet werden. Das Modell bezieht sich auf personenbezogene Informationen. Zur Schutzstufe 5 gehören medizinische Daten oder Daten über Straftaten.

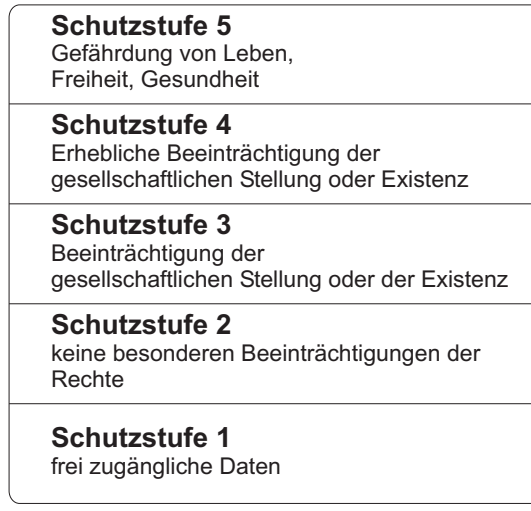


Abbildung 9.1: Das Schutzstufenmodell für Daten

Für sachbezogene, betriebswirtschaftliche Informationen könnte eine Einstufung wie folgt aussehen:

1. Frei zugängliche Informationen.
2. Informationen, deren Missbrauch keine besondere Beeinträchtigung der betrieblichen Funktion und Umweltbeziehung des Betriebes erwarten lässt.
3. Informationen, deren Missbrauch die betrieblichen Funktionen erheblich beeinträchtigt, die Umweltbeziehungen des Betriebes empfindlich stört oder das Ansehen des Betriebes beeinträchtigen kann.
4. Informationen, deren Missbrauch die finanz- oder marktwirtschaftliche Situation oder die Existenz des Betriebes erheblich beeinträchtigen kann.

Mit Kontrollfunktionen vor und nach den einzelnen Verarbeitungsschritten der Daten kann das Risiko des Datenmissbrauchs minimiert werden. Dabei gibt es eine Reihe von Kontrollfunktionen, wie das Absichern der räumlichen Zugänge (Zugangskontrolle), die der Datenbankentwickler sicherlich nicht beeinflussen kann.

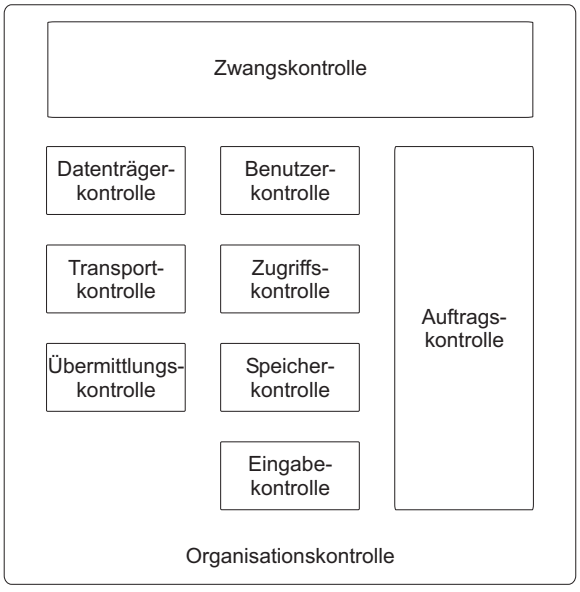


Abbildung 9.2: Kontrollfunktionen gegen Datenmissbrauch

Sehr wohl ist es für den Programmierer möglich, die mittlere Spalte von Benutzer- bis zur Eingabekontrolle zu beeinflussen und zu gestalten.

Kontrollfunktion	**Was soll verhindert werden?**	**Gestaltungsmerkmale**
Benutzerkontrolle	Unbefugte Personen haben Zugriff auf das System.	Festlegung von Benutzerberechtigungen, Mechanismen für die Identifikation und Authentifikation.
Zugriffskontrolle	Berechtigte Systembenutzer greifen auf Daten zu, für die sie keine Zugriffsberechtigung haben.	Zugriffsberechtigungen für Software und Daten, Mechanismen zur Identifikation und Authentifikation, Absicherung der IT.
Speicherkontrolle	Unbefugte Eingabe, Kenntnisnahme, Veränderung oder Löschung von Daten.	Festlegung der Befugnisse für die Eingabe, Überprüfung der Dateneingabe und Datenverarbeitung, kryptographische Verfahren zur Verschlüsselung, Mindestanforderungen an System- und Anwendungssoftware.
Eingabekontrolle	Unbefugtes Ändern von Daten.	Protokollierung des Benutzerkennzeichens.

Tabelle 9.1: Übersicht über Sicherheitskontrollfunktionen

9.2.2 Schutz vor Datenverlust durch Softwareversagen

Eine Risikoeinschätzung für das *Softwareversagen* ist meist sehr unzureichend. Schwachstellen bei hochkomplexen Systemen können auch durch umfangreiche Tests nicht vollständig ermittelt werden. Eingesetzte Standardsoftware stellt ebenfalls häufig eine Schwachstelle dar, weil sie aufgrund des Wettbewerbsdrucks schnell auf dem Markt erscheinen muss und deshalb nur die nötigsten Testphasen durchlaufen hat.

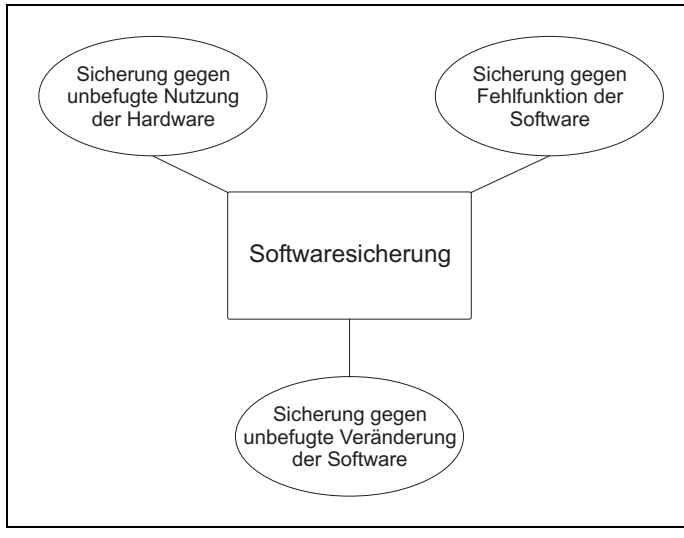


Abbildung 9.3: Übersicht zur Softwaresicherung

Zur Softwaresicherung gehören Maßnahmen, die unbefugte Nutzung und Veränderung zu unterbinden. Außerdem müssen Strategien entwickelt werden, welche die Funktion der Software sichern. Geeignete Maßnahmen wären Einsatz von *evaluierter Systemsoftware*, *Teststrategien*, *Protokollierung* und *Analyse* aller Fehler in System- und Anwendungssoftware und kontinuierliche Softwarepflege.

9.2.3 Schutz vor Datenverlust durch Hardwareversagen

Bei der Hardware sind insbesondere *Materialalterung* von System und Datenträgern als Schwachstelle zu nennen. *Geräteempfindlichkeit* gegenüber *Stromschwankungen* und vorübergehendes Entfernen von aktiven Komponenten im Netz bilden zusätzlich ein Risikopotential. Auch mit Zerstörung des Plattenspeichers durch *Kopflandungen* (Headcrash) muss gerechnet werden.

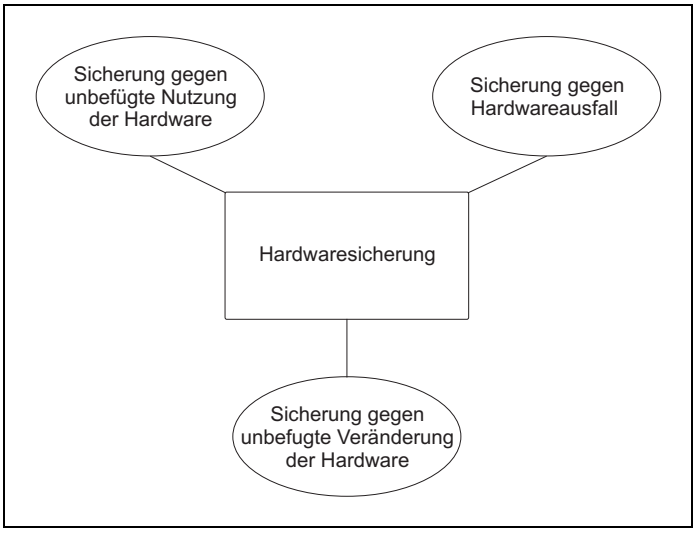


Abbildung 9.4: Übersicht zur Hardwaresicherung

Pflege und Wartung der Hardware kann einen Hardwareausfall vorbeugen oder ihn im Ernstfall minimieren. Häufig werden entsprechende Maßnahmen von geschultem Personal einer Wartungsfirma durchgeführt. Die Wartungsarbeiten können lokal oder bei entsprechenden Voraussetzungen auch per Fernwartung erledigt werden. Bei der Fernwartung ist darauf zu achten, dass »sensible« Daten mit notwendigen Schutzmechanismen vor unbefugter Datennutzung gesichert werden.

Bei der eingesetzten Hardware sollten Ersatzkapazitäten geschaffen werden. Besonders Schlüsselkomponenten im Netzwerk sollten doppelt ausgelegt oder zumindest schnell verfügbar sein. Bei Systemen, die eine Hochverfügbarkeit erfordern, ist *eine redundante Hardwareausstattung* unumgänglich. Beim Ausfall einzelner Komponenten müssen die anderen Bauelemente die Last des ausgefallenen übernehmen.

9.2.4 Konzeptlose Maßnahmen verursachen nur Kosten

Es ist gut Informationen, die per E-Mail verschickt werden, zu *verschlüsseln*. Es ist auch gut, den Web-Server mit einer *Firewall* abzusichern. Eine optische Spiegelung der Datenbestände kann das Sicherheitsgefühl zusätzlich steigern. Doch was nutzen all die technischen »Schutzmauern«, wenn Geschäftsbriefe nicht in den Schredder kommen, sondern in voller Pracht im Papierkorb landen? Es sind nicht immer Viren und Hacker, die für IT-Schäden verantwortlich sind. Zwei von drei IT-Angriffen gehen von eigenen Mitarbeitern aus und der wirtschaftliche Schaden, der dabei entsteht, ist immens. Sorglosigkeit und Unwissenheit gehen dabei Hand in Hand und lassen dadurch Firewalls, Antiviren- und Kryptographiesoftware »alt« aussehen.

Ohne ein umfassendes Sicherheitskonzept bleiben alle gutgemeinten Versuche – System und Daten zu schützen – nutzlos. Das Konzept ist abhängig von den Gegebenheiten im Unternehmen und von der Einstufung der Daten nach Schutzstufen. Einige Bereiche sollten aber in jedem Fall in dem Konzept zu finden sein.

- Einstufung der Daten,
- Berechtigungskonzepte für die IT-Nutzung,
- Administration und Rollenverteilung (insbesondere bei vernetzten IT-Systemen muss die Aufteilung der Administration klar geregelt sein),
- Identifikation und Authentifizierung der Benutzer,
- Datenhaltung und allgemeiner Umgang mit der IT,
- Festlegung von Hausstandards für IT-Komponenten,
- Gesicherte Anbindung an Fremdnetze,
- Sensibilisierung und Schulung der Administratoren und Benutzer.

Der folgende Maßnahmenkatalog ist ein Auszug aus den Schutzbestimmungen des BSI (Bundesamt für Informationssicherheit) für Hard- und Softwaremanagement.

Infrastruktur:

- Diebstahlsicherungen (optional).

Organisation:

- Datenträgerverwaltung,
- Nutzungsverbot nicht freigegebener Software,
- Überprüfung des Software-Bestandes,
- Regelung des Passwortgebrauchs,
- Betreuung und Beratung von IT-Benutzern (optional),
- Regelung für die Einrichtung von Benutzern / Benutzergruppen,
- Software-Abnahme- und Freigabe-Verfahren,
- Kontrolle der Protokolldateien,
- Einrichtung von Standardarbeitsplätzen,
- Datenschutzaspekte bei der Protokollierung,
- Bereithalten von Handbüchern,

- Strukturierte Datenhaltung,
- Sicheres Löschen von Datenträgern,
- Regelmäßige Kontrollen der organisatorischen IT-Sicherheitsmaßnahmen,
- Verhinderung ungesicherter Netzzugänge,
- Konzeption des IT-Betriebs,
- Fehlerbehandlung,
- Genehmigungsverfahren für IT-Komponenten,
- Sorgfältige Einstufung bzw. Umgang mit Informationen und Anwendungen.

System:

- Regelung der Mitnahme von Datenträgern und IT-Komponenten,
- Kontinuierliche Dokumentation der Informationsverarbeitung (insbesondere Administration)
- Richtlinien für die Zugriffs- bzw. Zugangskontrolle,
- Änderungsmanagement,
- Regelmäßige Kontrollen der technischen IT-Sicherheitsmaßnahmen,
- Sicherheitsvorgaben für die Nutzung von Standardsoftware,
- Vorbeugung gegen trojanische Pferde,
- Regelungen für den Einsatz von Fremdpersonal.

Personal:

- Einweisung des Personals in den sicheren Umgang mit IT.

Hardware/Software:

- Implementierung von Sicherheitsfunktionalitäten in der IT-Anwendung (optional),
- Test neuer Hard- und Software,
- Sorgfältige Durchführung von Konfigurationsänderungen,
- Software-Reinstallation bei Arbeitsplatzrechnern,
- Geeignete Auswahl von Authentifizierungs-Mechanismen (optional),
- Wahl geeigneter Datenformate,
- Restriktive Vergabe von Zugriffsrechten auf Systemdateien.

Kommunikation:

- Einsatz von Verschlüsselungsverfahren zur Netzkommunikation (optional),
- Bildung von Teilnetzen (optional),
- Vereinbarung über die Anbindung an Netze Dritter,
- Vereinbarung über Datenaustausch mit Dritten.

Notfallvorsorge:

- Redundante Kommunikationsverbindungen (optional).

Der Maßnahmenkatalog zeigt, dass der Großteil an Sicherheitsvorkehrungen nicht etwa in schützender Technologie zu finden ist, sondern bei vorbeugenden Maßnahmen im organisatorischen Bereich. Leider ist es in der Praxis so, dass diese Empfehlungen erst ernst genommen werden, wenn das Kind bereits in den Brunnen gefallen ist. Die aktuellen Statistiken über Schadensfälle lassen erkennen, dass zum Thema Sicherheit besonders in kleinen und mittleren Unternehmen Erkenntnismangel besteht. Auf der Begleit-CD finden Sie eine Checkliste, mit der organisatorische Sicherheitslücken in ihrem Unternehmen aufgedeckt werden können. Diese Liste wurde von den Mitarbeitern des Bundesamtes für Informationssicherheit erstellt.

9.3 Anmeldungsauthentifizierung

Die erste Hürde, die der Benutzer überwinden muss, wenn er auf Daten der Datenbank zugreifen will, ist die *Anmeldungsauthentifizierung*. Bei der Authentifizierung wird der Benutzer, der dafür sein Anmeldekonto verwendet, identifiziert. Das DBMS überprüft die Gültigkeit der vom Benutzer gemachten Angaben (Benutzerkonto und Kennwort) und stellt fest, ob eine Verbindung zum Datenbankserver hergestellt werden kann.

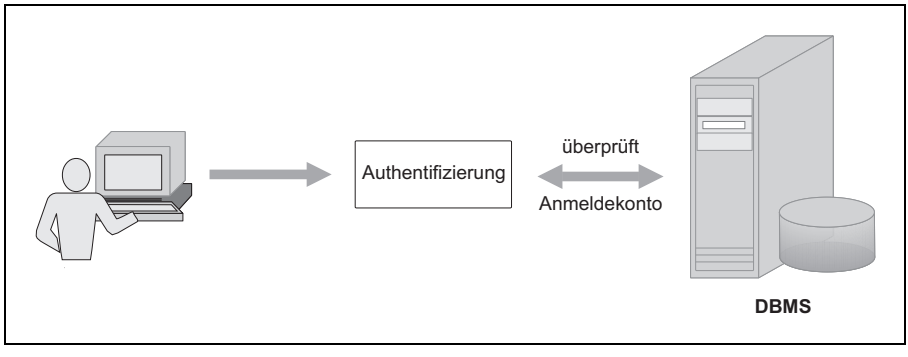


Abbildung 9.5: Anmeldeauthentifizierung bei Zugriff auf das DBMS

Die Anmeldeauthentifizierung ist die erste Sicherheitsebene auf der verhindert wird, dass unbefugte Benutzer sich Zugriff zum DBMS verschaffen. Dieser Sicherheitsmechanismus ist Standard und ist in nahezu jedem DBMS zu finden.

MS Access

Wird eine auf Access basierende Anwendungslösung im Netzwerk ausgeführt, sollten sowohl auf Datenbankebene als auch auf Benutzerebene die *Zugriffsrechte* abgefragt werden. Benutzer- und Gruppennamen sowie Kennwörter stehen in einer so genannten *Arbeitsgruppeninformationsdatei*. Bei der Installation von Access wird die Standarddatei SYSTEM.MDW angelegt. Mit dem *Arbeitsgruppen-Administrator* können Sie andere Arbeitsgruppeninformationsdateien erstellen und mit Access verknüpfen. Den *Arbeitsgruppen-Administrator* finden Sie in Ihrem Programmverzeichnis, in dem auch MS Access abgespeichert ist.

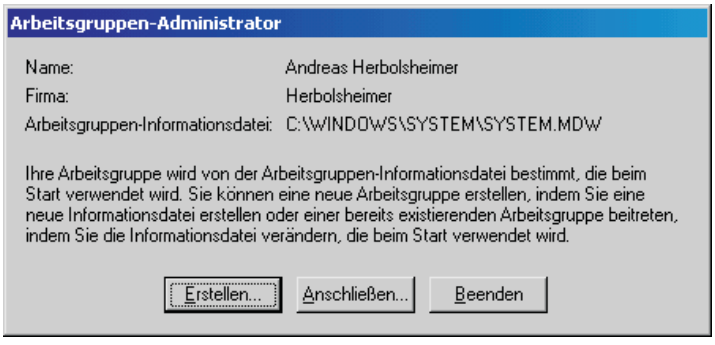


Abbildung 9.6: Der Arbeitsgruppen-Administrator von MS Access

Die Zuordnung eines Benutzers zu einer Arbeitsgruppeninformationsdatei wird in der Windows-Registrierung auf dem System des jeweiligen Benutzers gespeichert. Wie die einzelnen Datenbankbenutzer bzw. Gruppen angelegt werden, erfahren Sie im nächsten Abschnitt.

Wurden Arbeitsgruppen eingerichtet, so wird der Benutzer bereits beim Start von MS Access nach seinen Kenndaten gefragt. Diese müssen nicht zwangsläufig identisch sein mit den Kenndaten der Datenbankanwendung, was eine weitere Authentifizierung gegenüber dem System erfordern würde.

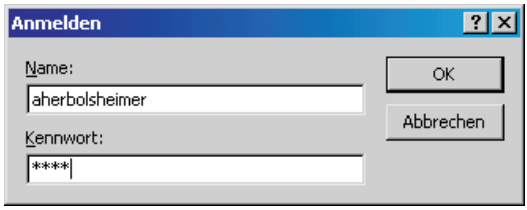


Abbildung 9.7: Anmeldung bei MS Access

MS SQL Server

Wollen Sie sich als Benutzer am SQL Server anmelden, dann kann das unter Umständen mit dem Benutzernamen und Kennwort geschehen, welches Sie für Windows-NT oder Windows 2000 verwendet haben. Beim SQL Server gibt es zwei *Authentifizierungsmodi*. Der Windows-Authentifizierungsmodus ermöglicht Benutzern mittels eines Microsoft Windows NT 4.0- oder Windows 2000-Benutzerkontos das Herstellen einer Verbindung. Außerdem gibt es einen *gemischten Modus*. Der gemischte Modus ermöglicht Benutzern mithilfe der *Windows-Authentifizierung* oder der *SQL Server-Authentifizierung* eine Verbindung zu einer Instanz von SQL Server herzustellen.

Eine Anmeldung kann wie folgt eingerichtet werden:

1. Erweitern der Konsolenstruktur, so dass der Ordner SICHERHEIT erscheint.
2. Klicken mit der rechten Maustaste auf BENUTZERNAMEN.
3. Auswählen von NEUER BENUTZERNAME aus dem Kontextmenü.
4. Im geöffneten Dialogfeld die Registerkarte ALLGEMEIN auswählen und die geforderten Eingaben vornehmen (siehe Abbildung 9.8).

PostgreSQL

Wenn eine Anwendung mit dem Datenbankserver verbunden wird, überprüft PostgreSQL die Kennung, ähnlich wie dies bei einer Unix-Anmeldung geschieht. Innerhalb der Datenbankumgebung wird anhand des Datenbankbenutzers auch der Zugriff auf diverse Datenbankobjekte zugelassen oder abgelehnt. Es gibt verschiedene Methoden für die Authentifizierung. Im Folgenden werden drei Authentifizierungsmethoden im Detail beschrieben:

Kennwortauthentifizierung

Datenbankkennwörter bei PostgreSQL sind verschieden von allen möglichen Kennwörtern des Betriebssystems. Gewöhnlich wird das Kennwort für jeden Datenbankbenutzer in einer Systemtabelle gespeichert. Kennwörter können mit den Befehlen der Abfragesprache `CREATE USER` und `ALTER USER` angelegt und verändert werden (siehe Kapitel 4). Wurde kein Kennwort gesetzt, so ist der gespeicherte Wert dafür der NULL-Wert und die Kennwortauthentifizierung fällt für diesen Benutzer aus.

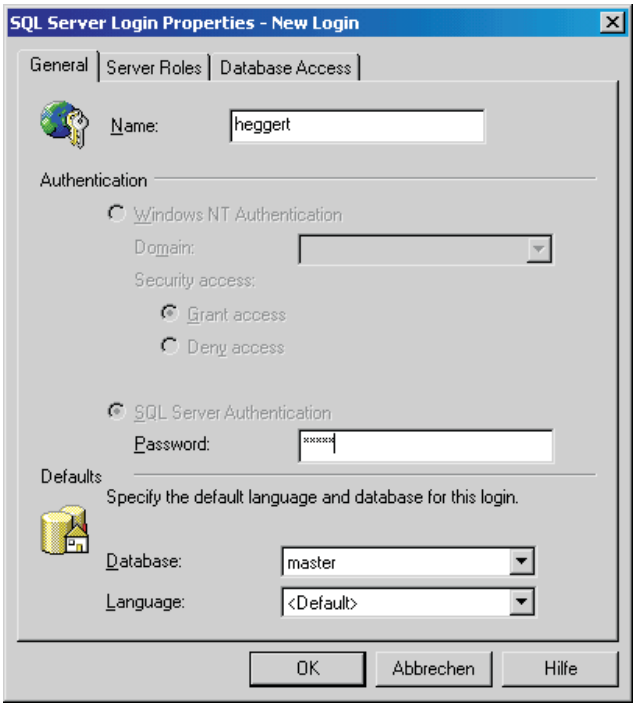


Abbildung 9.8: Einrichten eines neuen Login beim SQL Server

Kerberos-Authentifizierung

Kerberos ist ein industriekompatibles sicheres Authentifizierungssystem, das für verteilte Prozesse über Netzwerke verwendbar ist.

Ident-based Authentifizierung

Der Datenbankserver kommuniziert mit einem Server, der die Identifikationsmerkmale überprüft.

9.4 Einrichten und Verwalten der Datenbankbenutzer

Ein wichtiges Sicherheitskonzept bei DBMS sind Datenbankbenutzerkonten. *Benutzerkonten* definieren einen Benutzer in einer Datenbank und kontrollieren die Besitzer von Objekten und die Berechtigung zum Ausführen von Anweisungen. Benutzerkonten sind datenbankspezifisch. Datenbankbenutzer sind zu unterscheiden von den Benutzern des Betriebssystems. In einigen Fällen lässt sich das DBMS so steuern, dass der Benutzer des Betriebssystems auch für das DBMS übernommen werden kann.

9.4.1 Einrichten der Benutzer und Benutzergruppen

Bei der Installation der Datenbank werden standardmäßig *Benutzergruppen* und *Benutzer* angelegt, die beim ersten Anmelden am System verwendet werden können. Schließlich können Sie selbst Benutzername und Passwort für die einzelnen Benutzer und auch Gruppen von Benutzern definieren. Ein DB-Benutzername identifiziert einen Benutzer innerhalb einer Datenbank. Benutzer gleicher Art, d.h. mit gleicher Berechtigung, können als *GRUPPE* bzw. *ROLLE* zusammengefasst werden.

MS Access

Als Mitglied der Benutzergruppe *Administrator*, die automatisch bei der Installation von Access erzeugt wird, können Sie neue *Benutzergruppen* oder *Gruppenkonten* anlegen. Im Menü wählen Sie unter EXTRAS/ ZUGRIFFSRECHTE/BENUTZER- UND GRUPPENKONTEN...

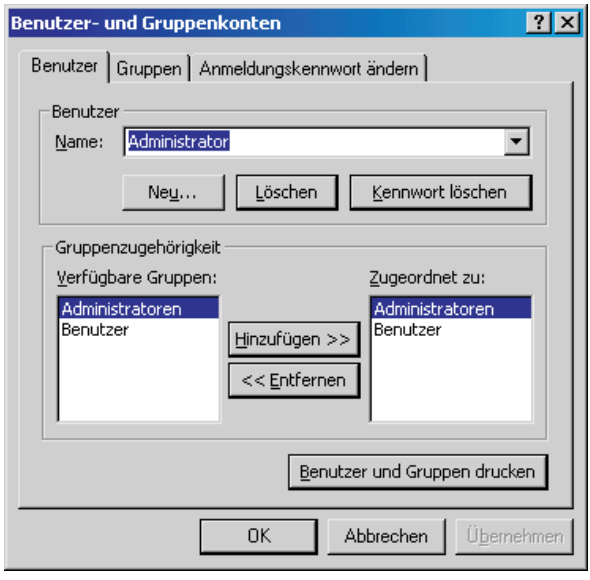


Abbildung 9.9: Benutzer- und Gruppenkonten in MS Access

Durch Anklicken der Befehlsschaltfläche NEU kann ein neuer Benutzer oder eine neue Benutzergruppe eingerichtet werden. Dafür muss ein Benutzername und eine persönliche ID spezifiziert werden, so wie dies in Abbildung 9.10 geschehen ist.

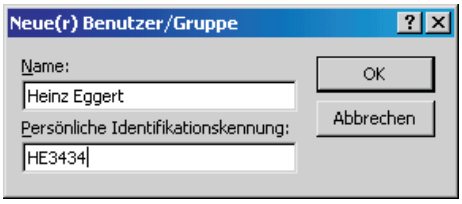


Abbildung 9.10: Neue Benutzerkennung in MS Access

Aus dem Namen und der ID wird vom System eine Sicherheits-ID erzeugt, die später weder eingesehen noch verändert werden kann. Das Passwort kann der Benutzer nur selbst vergeben. Er muss bei der ersten Anmeldung seinen Namen angeben und bei der Passwortabfrage keinen Eintrag vornehmen, dann kann er über EXTRAS/ ZUGRIFFSRECHTE / DATENBANKKENNWORT ZUWEISEN... sein Passwort definieren.

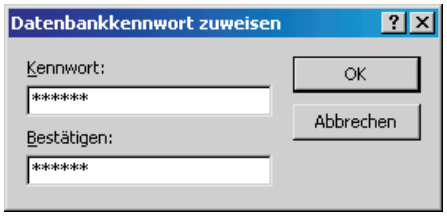


Abbildung 9.11: Vergabe eines Kennwortes in MS Access

Die Vergabe des Passwortes ist optional und kann später nicht mehr eingesehen werden. Ein vergessenes Passwort kann von einem Mitglied der Gruppe *Administrator* wieder gelöscht werden.

Dem neu angelegten Benutzer kann auch eine oder mehrere *Gruppenzugehörigkeiten* zugewiesen werden. Dafür muss einfach aus der Liste mit den verfügbaren Gruppen eine Selektion vorgenommen werden und die ausgewählten Gruppen zugewiesen werden (siehe Abbildung 9.9).

MS SQL Server

Beim SQL Server können Sie sich erstmalig über das Benutzerkonto *sa* anmelden und dann neue *Benutzer* mit Transact-SQL auf der Befehlszeilenebene einrichten (siehe Kapitel 4) oder im Enterprise Manager dialoggesteuert. Rufen Sie dazu im Enterprise Manager durch Klicken auf DATENBANKBENUTZER mit der rechten Maustaste in der ausgewählten Datenbank das Dialogfeld NEUER DATENBANKBENUTZER (New User) auf.

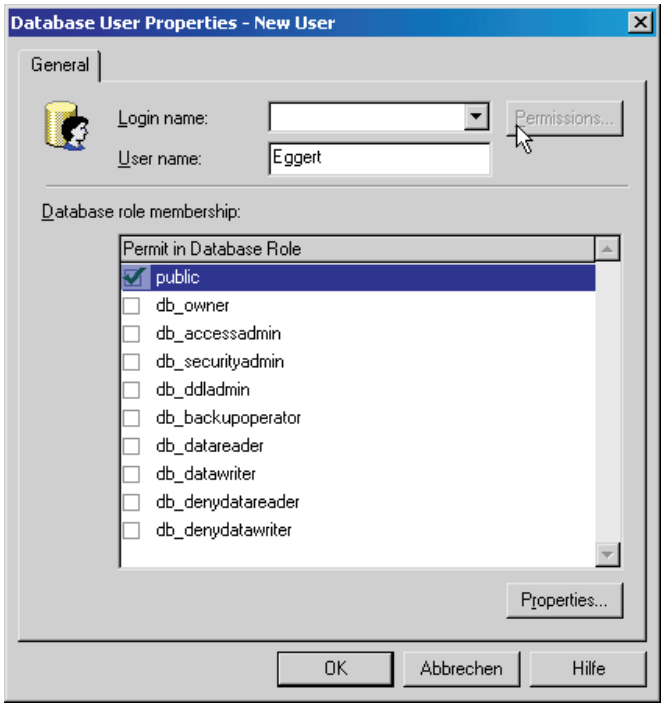


Abbildung 9.12: Einrichten eines neuen Datenbankbenutzers beim SQL Server

Aus der Auswahlliste kann eine bereits eingerichtete Authentifizierung gewählt werden und der Benutzername kann im dafür vorgesehenen Eingabefeld definiert werden. Auch hier ist es wieder möglich, dem Benutzer die Mitgliedschaft an *Rollen* zuzuweisen. Eine Authentifizierung über Windows-NT oder SQL-Server muss zuvor durchgeführt werden, denn die Angabe der Authentifizierung ist hier nicht optional. Jede Datenbank hat beim MS SQL Server genau einen *Datenbankbesitzer*, nämlich den Datenbankbenutzer, der die Datenbank erstellt hat. Der Datenbankbesitzer hat die Rechte eines Systemadministrators und kann anderen Benutzern die Berechtigung gewähren, auf die Datenbank zuzugreifen.

Mit *Datenbankrollen* können Berechtigungen an verschiedenen Datenbankobjekten für mehrere Benutzer zusammengefasst werden. Man könnte beispielsweise für die verschiedenen Abteilungen eines Unternehmens jeweils eine Rolle definieren. Jeder Mitarbeiter wäre dann Mitglied zumindest einer Rolle und hätte dieselbe Berechtigungen wie die anderen Mitglieder der Rolle. Es gibt feste Datenbankrollen, die bereits mit der Installation im DBMS vorhanden sind. Es gibt z.B. die feste Rolle *db_owner*, die den Mitgliedern uneingeschränkten Zugriff auf alle Datenbankobjekte gewährleistet.

Benutzerdefinierte Datenbankrollen lassen sich im Enterprise Manager einrichten. Sie müssen dazu in der betreffenden Datenbank die ROLLEN mit der rechten Maustaste anklicken und NEUE DATENBANKROLLE (New Role) wählen.

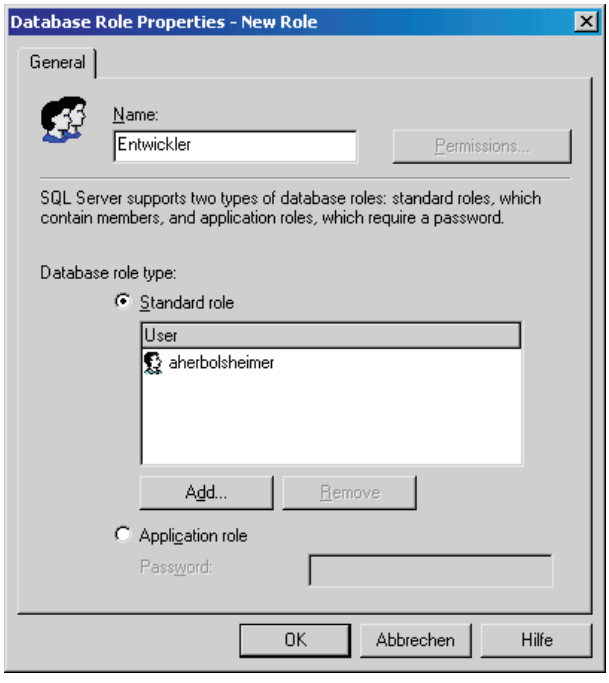


Abbildung 9.13: Einrichten einer neuen Datenbankrolle

Es gibt zwei Typen bei den benutzerdefinierten Rollen:

- Standardrollen,
- Anwendungsrollen.

Bei Anwendungsrollen kann die Sicherheitssteuerung über die aufgesetzte Anwendung realisiert werden. Standardrollen entsprechen den Gruppen in anderen DBMS.

PostgreSQL

Bei PostgreSQL sind die Namen der Datenbankbenutzer global gültig, d.h. über alle vorhandenen Datenbanken. Mit der CREATE USER-Anweisung kann ein neuer Benutzer eingerichtet werden:

Beispiel 9.1:

```
CREATE USER eggert
```

Beim ersten Anmelden muss der vordefinierte Benutzername verwendet werden (z.B. postgres). Sie starten also den Postmaster mit der Anweisung:

```
rcpostgres start
```

Anschließend kann man sich via *su* als Benutzer *postgres* an der Datenbank anmelden.

```
su - postgres
```

Mit der Anweisung aus Beispiel 9.1 legen Sie weitere Benutzer an. Der Benutzername muss den Regeln für Bezeichner entsprechen. Jeder Benutzer kann einige Attribute haben, welche die Berechtigungen des Benutzers beschreiben. Mit einer zusätzlichen Passwort-Klausel lässt sich das Passwort definieren:

Beispiel 9.2:

```
CREATE USER eggert WITH PASSWORD 'xxxxx'
```

Benutzergruppen können ebenfalls über SQL-Anweisungen angelegt werden.

Beispiel 9.3:

```
CREATE GROUP entwickler
```

Die Anweisung in Beispiel 9.3 erstellt eine logische *Benutzergruppe* mit dem Namen *entwickler*.

9.4.2 Löschen der Benutzer und Gruppen

Den einmal angelegten Datenbankbenutzern können Rechte zugewiesen werden, sie können Mitglieder anderer Benutzergruppen werden, verwaiste Benutzergruppen müssen schließlich gelöscht werden. Auch das Löschen der Benutzer oder der Benutzergruppen kann über SQL-Anweisungen, aber auch über die graphische Benutzeroberfläche dialoggesteuert geschehen. Letzteres soll wieder für MS Access und den MS SQL Server gezeigt werden. Bei PostgreSQL werden wir dann auf die entsprechende SQL-Anweisung zurückgreifen.

MS Access

Benutzer- und Gruppenkonten können auf die gleiche Art und Weise wieder gelöscht werden, wie Sie auch zuvor erstellt wurden. Im Menü ist dazu EXTRAS zu wählen und über ZUGRIFFSRECHTE und GRUPPEN- UND BENUTZERKONTEN... wird das Dialogfeld BENUTZER- UND GRUPPENKONTEN geöffnet. Wollen Sie eine Gruppe löschen, dann wählen Sie die Registerkarte GRUPPEN und löschen Sie dort die betreffende Gruppe (siehe Abbildung 9.14). Soll nur ein Benutzer entfernt werden, dann kann dies auf der Registerkarte BENUTZER erledigt werden.

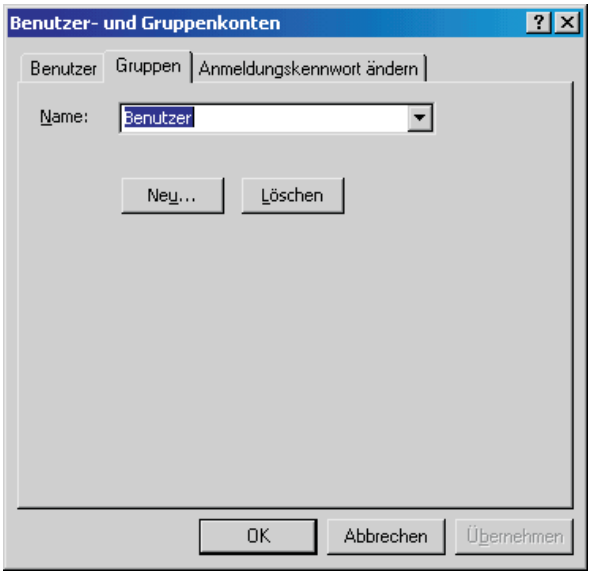


Abbildung 9.14: Löschen eines Benutzerkontos in MS Access

MS SQL Server

Ein Benutzername kann beim SQL Server nur dann entfernt werden, wenn dieser keine Objekte mehr besitzt. Ist dies nicht der Fall, müssen die Objekte zuerst entfernt oder einem anderen Benutzer übertragen werden. Hat der Benutzer kein Objekt in der Datenbank, dann kann er im *Enterprise Manager* gelöscht werden:

1. Erweitern der Konsolenstruktur und Auswählen der Datenbank.
2. Klicken auf BENUTZERNAMEN.
3. Im rechten Fensterbereich den zu löschenden Benutzer auswählen.
4. Drücken auf Entf oder Klicken der rechten Maustaste und Auswahl von LÖSCHEN (Delete) (siehe Abbildung 9.15).
5. Sicherheitsabfrage mit JA bestätigen.

Durch Entfernen eines Benutzers aus einer SQL Server-Datenbank werden automatisch die Berechtigungen entfernt, die für den Benutzer oder die Gruppe definiert wurden. So wird verhindert, dass der Benutzer die Datenbank unter dem alten Sicherheitskonto verwendet. Die Berechtigungen müssen nicht separat entfernt werden. Wollen Sie eine *Rolle* aus der Datenbank entfernen, ist die Vorgehensweise nahezu identisch mit der beschriebenen.

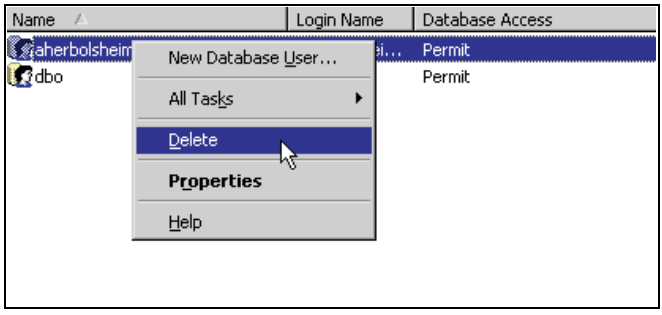


Abbildung 9.15: Löschen eines Benutzers beim SQL Server

PostgreSQL

Ein Benutzer kann mit dem *DROP USER*-Befehl aus dem System entfernt werden:

Beispiel 9.4:

```
DROP USER  eggert
```

Mit `DROP USER` werden keine Datenbankobjekte entfernt. Besitzt der Benutzer noch Objekte in der Datenbank, so kann er nicht gelöscht werden. Gruppen werden mit der DROP GROUP-Anweisung gelöscht.

Beispiel 9.5:

```
DROP GROUP entwickler
```

Die Mitglieder dieser Gruppe werden nicht gelöscht.

9.5 Berechtigungsprüfung auf Ebene der Ressourcen

Hat sich der Datenbankbenutzer ordnungsgemäß angemeldet, heißt dies noch nicht, dass er damit automatisch einen »Freifahrtschein« für alle denkbaren Aktivitäten an der Datenbank hat. Das DBMS überprüft bei jeder Aktion, ob der Benutzer die Berechtigung zum Lesen oder Schreiben hat und ob er sie für das betreffende Objekt hat. In einem Benutzerkonto sind die Berechtigungen aufgelistet, die der jeweilige Benutzer für die einzelnen Objekte der Datenbank hat. Wie Objektberechtigungen in MS Access, SQL Server und PostgreSQL vergeben werden, soll an dieser Stelle demonstriert werden.

MS Access

In Access lassen sich für Tabellen, Abfragen, Formulare, Berichte, Module und Makros Benutzer- und Gruppenberechtigungen vergeben. Nur Mitglieder der Benutzergruppe Administratoren können Änderungen an den Benutzerkonten vornehmen. Die Zugriffsrechte können im Dialogfeld Benutzer- und Gruppenberechtigung festgelegt werden:

1. Im Menü EXTRAS Zugriffsrechte wählen.
2. Das Dialogfeld BENUTZER- UND GRUPPENBERECHTIGUNG öffnen.
3. Registerkarte BERECHTIGUNGEN auswählen.
4. Benutzer- oder Gruppenname selektieren.
5. Objekt markieren.
6. Berechtigung zuweisen.

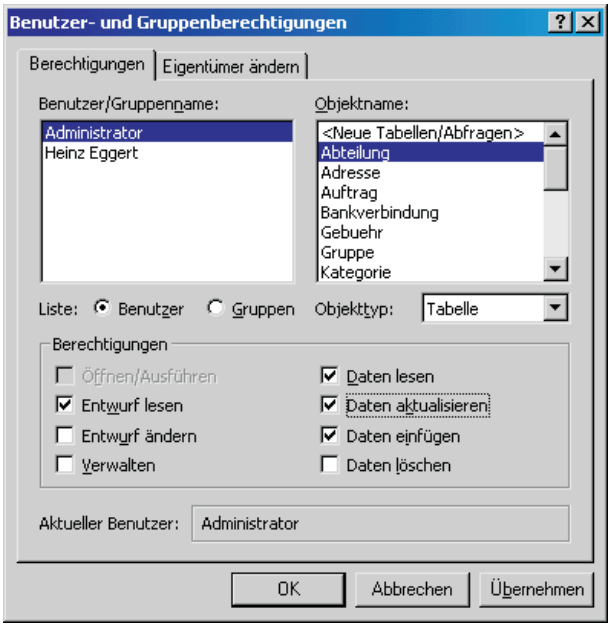


Abbildung 9.16: Benutzer- und Gruppenberechtigung in MS Access

Durch eine Zuordnung der einzelnen Benutzer zu einer Benutzergruppe lässt sich die Verwaltung von Berechtigungen vereinfachen. Die Zugriffsrechte werden dann den Gruppen zugewiesen und die Benutzer werden dann Mitglied bei der Gruppe mit den entsprechenden Rechten.

MS SQL Server

Jedes Objekt einer Datenbank hat einen Besitzer. Der Besitzer hat sämtliche Rechte an diesem Objekt und kann diese an andere Datenbankbenutzer weitergeben. Wenn der Besitzer nur bestimmten Benutzern den Zugriff auf das Objekt erteilen möchte, kann er nur diesen Benutzern *Berechtigungen* erteilen. Für unterschiedliche Aktionen wie `INSERT`, `DELETE` oder `SELECT` muss dem Benutzer erst die Berechtigung an der Tabelle oder Sicht zugewiesen werden. Es ist möglich, nur für Objekte in der aktuellen Datenbank Berechtigungen für Benutzerkonten zu erteilen. Wenn ein Benutzer Berechtigungen für Objekte in einer anderen Datenbank benötigt, erstellen Sie das Benutzerkonto in der anderen Datenbank oder erteilen Sie dem Benutzerkonto Zugriff auf die aktuelle Datenbank und auf die andere Datenbank. Mit den angegebenen Schritten kann im Enterprise Manager eine Rechtevergabe vorgenommen werden:

1. Erweitern der Konsole, bis die gewünschte Datenbank sichtbar ist.
2. Klicken auf BENUTZER oder ROLLE.
3. Im rechten Fensterbereich auf den gewünschten Benutzer oder die gewünschte Gruppe doppelt klicken.
4. Klicken auf BERECHTIGUNGEN VERWALTEN.
5. Im Dialogfeld OBJEKTEIGENSCHAFTEN (Properties) können die Berechtigungen vergeben werden (siehe Abbildung 9.17).

Selbstverständlich können Sie auch mit der SQL-Anweisung `GRANT` eine Vergabe von Berechtigungen durchführen. Wie dies funktioniert, wurde bereits in Kapitel 4 gezeigt. Mit `REVOKE` können Berechtigungen entzogen werden. Im Enterprise Manager wird eine Berechtigung entzogen, indem im Dialogfeld OBJEKTEIGENSCHAFTEN für das Objekt und die Aktion ein X gesetzt wird. Berechtigungen können auch für gespeicherte Prozeduren, Trigger oder einzelne Spalten von Tabellen und Sichten vergeben werden.

PostgreSQL

Nur der Besitzer des Datenbankobjektes oder der *Superuser* kann Berechtigungen an andere Benutzer erteilen. Es gibt vier verschiedene Berechtigungen für ein Datenbankobjekt:

- READ (für SELECT-Anweisungen),
- APPEND (für INSERT-Anweisungen),
- WRITE (für UPDATE- und DELETE-Anweisungen),
- RULE (die Berechtigung eine Regel für eine Tabelle zu erstellen).

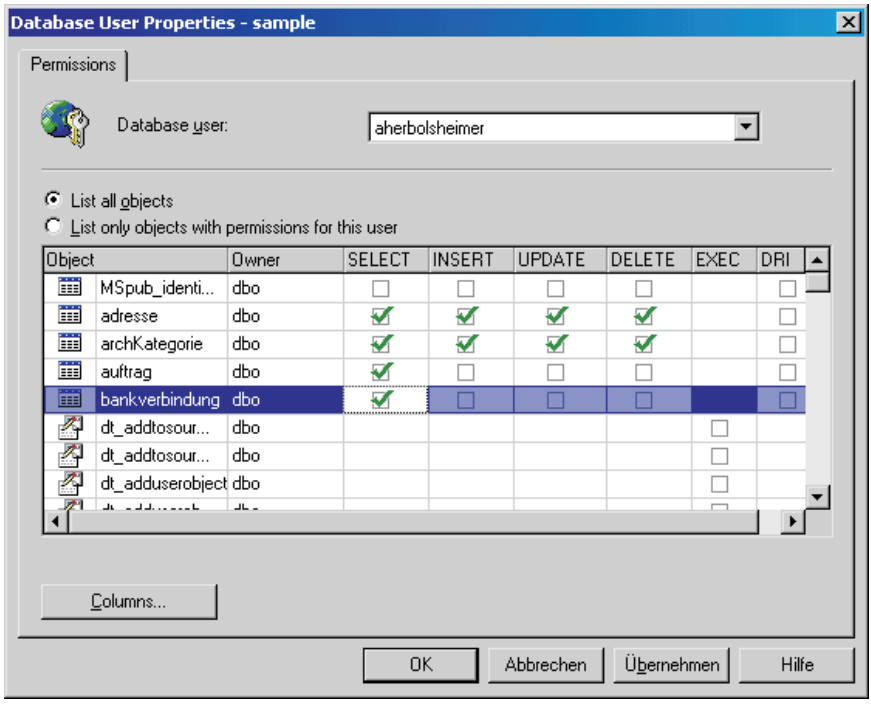


Abbildung 9.17: Vergabe der Berechtigungen beim SQL Server

Mit den SQL-Anweisungen `GRANT` und `REVOKE` werden die Berechtigungen zugeteilt und entzogen.

Beispiel 9.6:

```
GRANT SELECT ON service TO GROUP entwickler;
```

Die Benutzergruppe *entwickler* erhält im Beispiel 9.6 die Berechtigung, die Tabelle *service* zu lesen. Soll dem Benutzer nur die Leseberechtigung für einige Spalten der Tabelle erteilt werden, so muss dies mit einer Sicht gelöst werden. Es wird eine Sicht mit den gewünschten Spalten definiert und der Benutzer bekommt das Leserecht für die Sicht und nicht für die Basistabelle.

9.6 Verschlüsselung von Informationen

Die *Verschlüsselung* ist eine Methode, Informationen vertraulich zu behandeln, indem Daten in eine unlesbare Form geändert werden. Durch die Verschlüsselung soll sichergestellt werden, dass die Informationen nicht von unbefugten Dritten gelesen werden. Die *Entschlüsselung* ist der Prozess, die verschlüsselten Daten in die ursprüngliche Form zurück zu ändern, damit sie von authorisierten Benutzern angezeigt werden können.

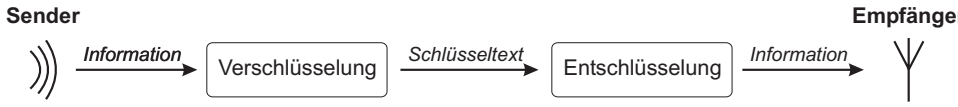


Abbildung 9.18: Schema der Datenverschlüsselung

Symmetrische Verschlüsselung

Das symmetrische Verfahren (Private Key-Verfahren) basiert auf einem gemeinsamen Schlüssel, der vom Sender und Empfänger vereinbart wird. Dieser Schlüssel wird sowohl zum Ver- als auch zum Entschlüsseln der Information verwendet. Vor dem Austausch der Information muss dieser Schlüssel über einen sicheren Kanal zwischen den Kommunikationspartnern ausgetauscht werden. Bei vielen Teilnehmern, die am Kommunikationsvorgang beteiligt sind, führt dieses Verfahren zu einer großen Anzahl an Schlüsseln.

Asymmetrische Verfahren

Im Gegensatz zum symmetrischen, benutzt das asymmetrische Verfahren für die Verschlüsselung und die Entschlüsselung verschiedene Schlüssel. Es gibt einen so genannten öffentlichen Schlüssel (Public Key), der zum Verschlüsseln berechtigt und einen geheimen Schlüssel, mit dem die Information entschlüsselt werden kann. Das asymmetrische Verfahren bietet gegenüber dem symmetrischen den Vorteil, dass die Anzahl der geheimen Schlüssel, über die die Partner verfügen, erheblich geringer ist (siehe Abbildung 9.19).

Sollen sensible Daten über ein Transportsystem übertragen werden, das keine ausreichende Sicherheit gegen Vertraulichkeitsverlust, Integrität oder Verfügbarkeit bietet, dann sollten die Daten verschlüsselt werden.

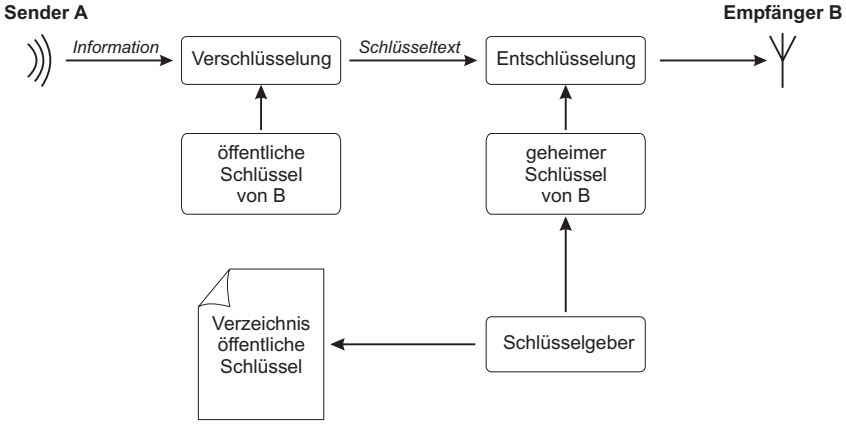


Abbildung 9.19: Asymmetrische Verfahren der Datenverschlüsselung

Bei MS Access besteht die Möglichkeit, auf Ebene der Datenbank Daten zu verschlüsseln. Mit einer Datenbankverschlüsselung wird verhindert, dass die Inhalte mit anderen Programmen als Access ausgewertet werden. Andernfalls könnte eine Datenbankdatei z.B. mit einem Texteditor geöffnet und Daten so gelesen werden. Eine Verschlüsselung der Datenbank verschlechtert allerdings die Leistung des laufenden Betriebs.

Sie können die Datenbank verschlüsseln, indem Sie im Menü EXTRAS / SICHERHEIT / DATENBANK VER-/ENTSCHLÜSSELN wählen. Im geöffneten Dialogfeld (siehe Abbildung 9.20) können Sie nun die zu verschlüsselnde Datenbankdatei auswählen und einen neuen Namen für die verschlüsselte Datenbankdatei vergeben. Das Original bleibt unverschlüsselt (siehe Abbildung 9.20).

Ein allgemein verbreitetes und anerkanntes Werkzeug zur Verschlüsselung von Daten ist PGP (Pretty Good Privacy). PGP ist ein Verschlüsselungsprogramm für elektronische Post und Dateien aller Art. Es ist neuerdings auch als Plug-Ins für Standardsoftware erhältlich. Bei der Verschlüsselung wird das bereits beschriebene asymmetrische Verfahren mit öffentlichen und geheimen Schlüsseln verwendet (siehe Abbildung 9.21).

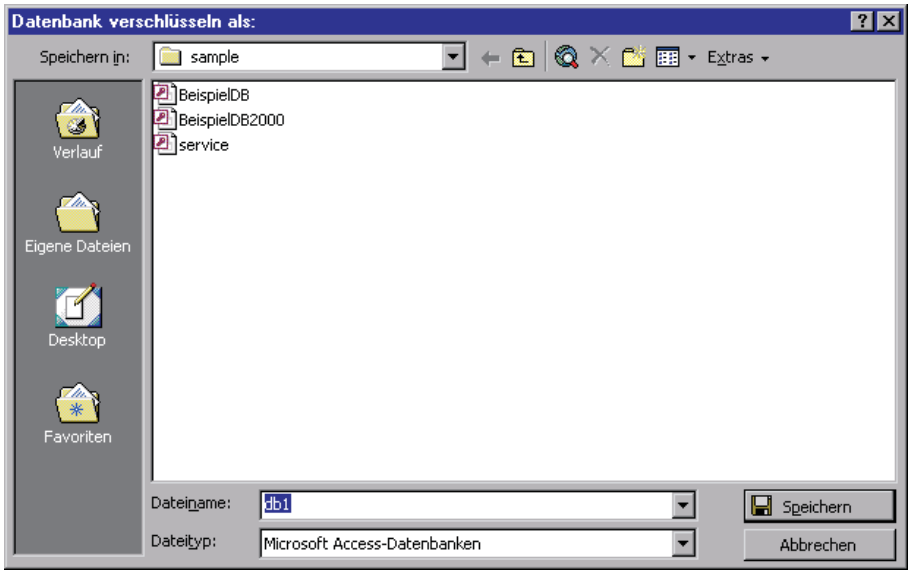


Abbildung 9.20: Datenbank verschlüsseln in MS Access

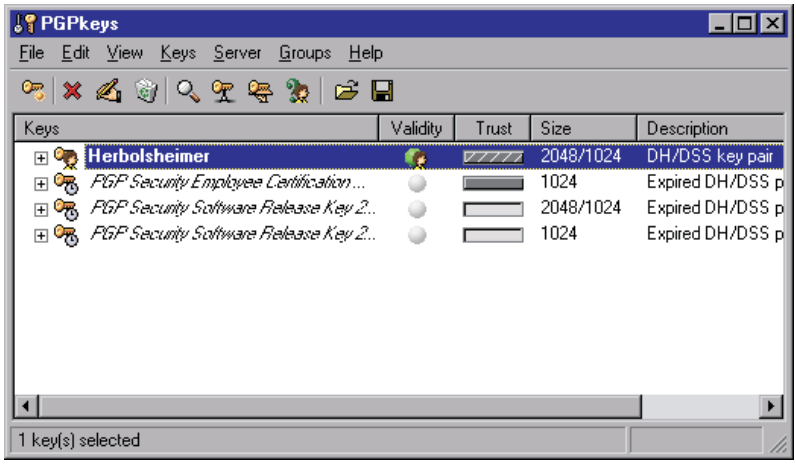


Abbildung 9.21: Programmoberfläche von Pretty Good Privacy

9.7 Datenbank sichern

Die Strategie für die Datensicherung sollte Bestandteil des gesamten Sicherheitskonzeptes sein. Darin müssen zumindest die folgenden Punkte festgehalten sein:

- für die Sicherung verantwortlicher Personenkreis,
- welche Daten werden gesichert,
- Sicherungsmedium,
- verwendeter Sicherungsmodus,
- Zeitpunkt der Sicherung,
- Überprüfen und Lagern des Sicherungsmediums.

Erst wenn diese Punkte geklärt sind, ist eine effektive Datensicherung möglich. Der erste Punkt wird relativ schnell beantwortet sein. Üblicherweise ist der Datenbankadministrator für die Sicherung der Daten zuständig. Etwas komplizierter wird es, wenn es mehrere Administratoren gibt oder der Administrator nur an bestimmten Tagen in der Firma ist. In diesen Fällen ist es wichtig, dass mehrere Personen in die Sicherungsstrategie eingeweiht werden und Zuständigkeiten klar definiert sind.

Von großer Wichtigkeit ist die Entscheidung, welche Daten gesichert werden sollen. Es müssen alle Daten gesichert werden, die die Datenbank wieder in den Zustand bringen, der zum Zeitpunkt der Sicherung bestand. Dazu gehören neben den *Benutzerdaten* auch die *Systemdaten*.

Als *Sicherungsmedium* können Bänder, Festplatten oder auch Named Pipes verwendet werden. Soll eine lokale Sicherung auf der Festplatte durchgeführt werden, so sollte dies nicht auf derselben Festplatte geschehen, auf der sich die Datenbank befindet. Bei einem *Festplattencrash* wäre die Sicherung genauso betroffen wie die zu sichernde Datenbank selbst.

Mit Hilfe von Named Pipes können Datenbanksicherungen über die Software von Drittanbietern durchgeführt werden.

Beim *Sicherungsmodus* geht es um den verwendeten Algorithmus, der zur Datensicherung herangezogen wird. In der Tabelle 9.2 werden einige Modi vorgestellt, die beim MS SQL Server oder bei PostgreSQL angeboten werden.

Name	Beschreibung	Datenbank
Vollständige Datenbanksicherung	Vollständige Kopie der Datenbank	MS SQL Server
Differentielle Datenbanksicherung	Nur die Änderungen seit der letzten vollständigen Datenbanksicherung werden berücksichtigt.	MS SQL Server
Transaktionsprotokollsicherung	Sicherung des Protokolls und damit Sicherung der Transaktionen.	MS SQL Server
Dateisicherung	Einzelne Dateien oder Dateigruppen werden gesichert.	MS SQL Server
SQL Dump	In einer Textdatei werden SQL-Anweisungen gespeichert, die bei der Wiederherstellung dafür sorgen, dass die Datenbank in dem Zustand aufgebaut wird, in dem sie sich zum Zeitpunkt der Datenbanksicherung befand.	PostgreSQL
File System Level	Äquivalent zur Dateisicherung werden hier die Dateien gesichert, welche relevante Daten enthalten.	PostgreSQL

Tabelle 9.2: Sicherungsmodi der DBMS

Die Frage, wann und wie häufig eine Datenbanksicherung durchgeführt werden soll, hängt von mehreren Faktoren ab. Ein wichtiger zu berücksichtigender Faktor ist beim Datenbestand selbst zu suchen. Finden häufiger Transaktionen statt, bei denen die Daten verändert werden, muss selbstverständlich auch öfter eine Datenbanksicherung vorgenommen werden. Außerdem sollte eine Datenbanksicherung nicht den laufenden Betrieb beeinträchtigen. Diese Forderung kann dazu führen, dass nur eine geringe Zeitspanne zur Sicherung zur Verfügung steht (meist ist das ein Zeitintervall in der Nacht). In diesem Fall sollte man mit einer Strategie arbeiten, die tägliche Sicherung der Benutzerdatenbank (oder zumindest Teile davon) erstellt und wöchentlich die Systemdaten sichert. In einer Testphase sollte die Zeit, die zur Datenbanksicherung erforderlich ist, ebenso ermittelt werden, wie die benötigten Ressourcen auf den Sicherungsmedien. Bestandteil der Testläufe sollte auch eine Wiederherstellung der Datenbank auf einem Testserver unter Verwendung der Sicherung sein. Die Effektivität einer guten Strategie erweist sich erst in der praktischen Umsetzung und diese Sicherheit sollte man sich nicht erst im Notfall bestätigen lassen.

9.7.1 Praktische Schritte zum Backup

Nach all den Vorüberlegungen wird in diesem Abschnitt Schritt für Schritt die praktische Umsetzung einer Datenbanksicherung für den MS SQL Server und PostgreSQL beschrieben.

MS SQL Server

1. Sicherungsmedium erstellen:
   - Erweitern der Konsolenstruktur, bis unter VERWALTUNG der Eintrag SICHERUNG erscheint.
   - Mit der rechten Maustaste SICHERUNG anklicken und im Kontextmenü NEUES MEDIUM auswählen.
   - Im geöffneten Dialogfeld EIGENSCHAFTEN VON SICHERUNGSMEDIUM (Backup Device Properties) kann ein logischer Name für das Medium vergeben und der Pfad für die Sicherung vorgegeben werden (siehe Abbildung 9.22).
   - Klicken auf OK, um das Medium zu erstellen.

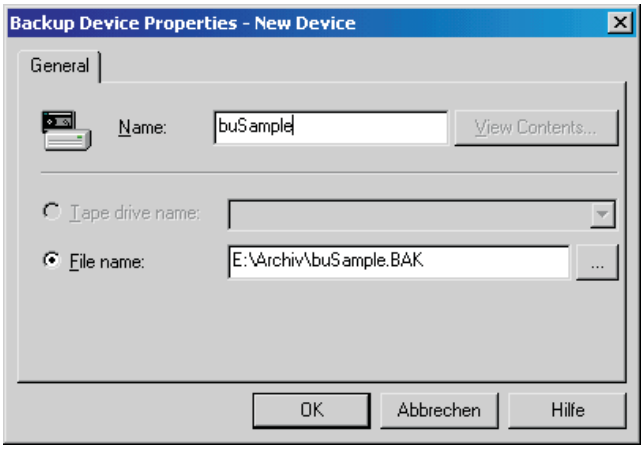


Abbildung 9.22: Namensangaben für ein neues Backup

2. Backup erstellen:
   - Erweitern der Konsolenstruktur, bis unter Verwaltung der Eintrag SICHERUNG erscheint.
   - Mit der rechten Maustaste SICHERUNG anklicken und im Kontextmenü DATENBANK SICHERN auswählen.

- Im Dialogfeld DATENBANK SICHERN wählen Sie die Registerkarte ALLGEMEIN aus.
- Die Datenbank ist zu wählen und ein logischer Name für die Sicherung und eine Beschreibung vorzugeben.
- Der Sicherungsmodus ist auszuwählen.
- Das Zielmedium muss, wie in Abbildung 9.23 zu sehen, angegeben werden (siehe Punkt 1 – »Sicherungsmedium erstellen«).

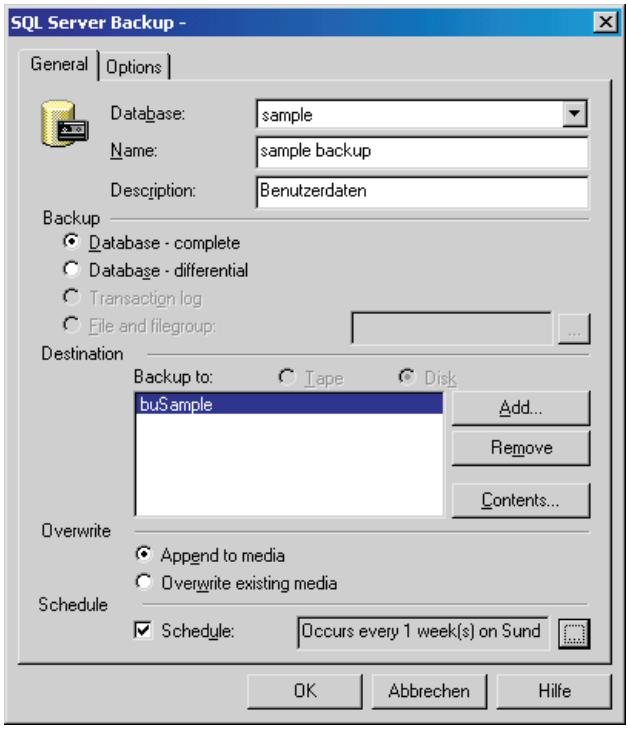


Abbildung 9.23: Einrichten des Zielmediums für das Backup

- Festlegen, ob Sicherung an das Medium angehängt wird oder vorhandenes Medium überschrieben wird.
- Optionale Definition eines Terminplans (siehe Abbildung 9.25).
- Vorgaben mit OK bestätigen (siehe Abbildung 9.25).

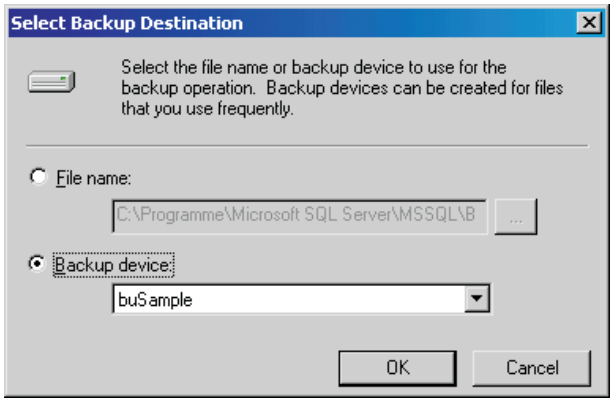


Abbildung 9.24: Zielmedium für Backup

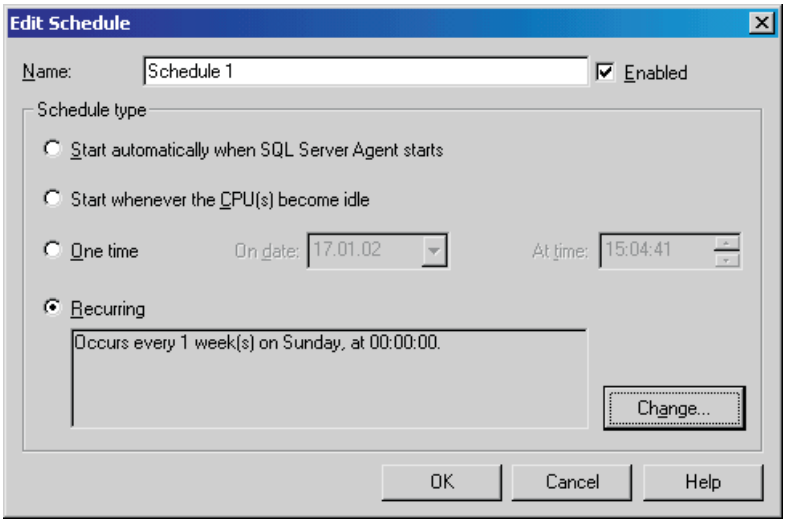


Abbildung 9.25: Erstellen eines Terminplans für die Datensicherung

Die Sicherung wird nicht sofort durchgeführt, sondern gemäß der Angabe im Terminplan. Dieses ist nur eine Vorgehensweise von mehreren. So ist es auch denkbar, die Datenbanksicherung mit einem Assistenten (Wizard) zu planen. Diese Vorgehensweise ist jedoch überwiegend selbsterklärend und wird deshalb nicht weiter erläutert. Außerdem kann die Sicherung per Transact-SQL erfolgen. Die Syntax für die Anweisung sieht wie folgt aus:

```
BACKUP DATABASE { database_name | @database_name_var }
TO < backup_device > [ ,...n ]
[ WITH
```

```
    [ BLOCKSIZE = { blocksize | @blocksize_variable } ]
    [ [ , ] DESCRIPTION = { 'text' | @text_variable } ]
    [ [ , ] DIFFERENTIAL ]
    [ [ , ] EXPIREDATE = { date | @date_var }
        | RETAINDAYS = { days | @days_var } ]
    [ [ , ] PASSWORD = { password | @password_variable } ]
    [ [ , ] FORMAT | NOFORMAT ]
    [ [ , ] { INIT | NOINIT } ]
    [ [ , ] MEDIADESCRIPTION = { 'text' | @text_variable } ]
    [ [ , ] MEDIANAME = { media_name | @media_name_variable } ]
    [ [ , ] MEDIAPASSWORD = { mediapassword | @mediapassword_variable } ]
    [ [ , ] NAME = { backup_set_name | @backup_set_name_var } ]
    [ [ , ] { NOSKIP | SKIP } ]
    [ [ , ] { NOREWIND | REWIND } ]
    [ [ , ] { NOUNLOAD | UNLOAD } ]
    [ [ , ] RESTART ]
    [ [ , ] STATS [ = percentage ] ]
]
```

Beschreibung:

database_name | *@database_name_var*
: Logischer Name der Datenbank, angegeben als Zeichenfolge oder Variable.

backup_device
: Logisches oder physisches Sicherungsmedium.

blocksize | *@blocksize_variable*
: Legt die physische Blockgröße in Byte fest.

'text' | *@text_variable*
: Der Beschreibungstext wird als Zeichenfolge oder als Konstante übergeben.

DIFFERENTIAL
: Der Sicherungsmodus Differential wird gewählt.

EXPIREDATE = date | *@date_var*
: Angabe eines Datums, zu dem der Sicherungssatz abläuft und überschrieben werden kann.

RETAINDAYS = days | *@days_var*
: Angabe einer Zeitspanne in Tagen, nach der der Datensatz überschrieben werden darf.

PASSWORD = password | *@password_variable*
: Es wird ein Kennwort für den Sicherungssatz festgelegt. Bei Angabe eines Kennwortes, wird dieses für die Wiederherstellung benötigt.

FORMAT
Formatierung des Datenträgers vor der Sicherung in der Form, dass ein vorhandener Medienvorspann überschrieben wird.

NOFORMAT
gibt an, dass der Medienvorspann nicht auf alle Datenträger geschrieben werden soll, die für diesen Sicherungsvorgang verwendet werden, und dass das Sicherungsmedium nur dann neu geschrieben wird, wenn INIT angegeben ist.

INIT | NOINIT
INIT gibt an, dass alle Sicherungssätze überschrieben werden sollen, während der Medienvorspann erhalten bleibt. NOINIT zeigt an, dass der Sicherungssatz an das Ende des angegebenen Datenträger- oder Bandmediums angefügt wird, wobei die vorhandenen Sicherungssätze erhalten bleiben.

MEDIADESCRIPTION = 'text' | @text_variable
Es kann eine Beschreibung des Mediensatzes vorgenommen werden.

MEDIANAME = media_name | @media_name_variable
Namensangabe für den Sicherungsmediensatz.

MEDIAPASSWORD = mediapassword | @mediapassword_variable
legt das Kennwort für den Mediensatz fest.

NAME = backup_set_name | @backup_set_name_var
Name des Sicherungssatzes.

NOSKIP
Weist die BACKUP-Anweisung an, das Ablaufdatum aller Sicherungssätze auf den Medien zu prüfen, bevor diese überschrieben werden dürfen.

SKIP
deaktiviert die Prüfung von Ablaufdatum und Namen der Sicherungssätze.

NOREWIND
Das Band wird nach dem Sicherungsvorgang offen gehalten.

REWIND
gibt an, dass SQL Server das Band freigibt und zurückspult.

RESTART
Ein unterbrochener Sicherungsvorgang wird neu gestartet.

STATS = percentage
Fortschrittsanzeige bei Datensicherung bei jedem Prozentanteil (percentage).

Der Transact-SQL Befehl für das Sichern des Transaktionsprotokolls *BACKUP LOG* verwendet in etwa dieselben Angaben und wird deshalb nicht weiter erläutert. Wie Sie diese Anweisung z.B. im Query Analyzer verwenden können, wird im Bild 9.26 gezeigt.

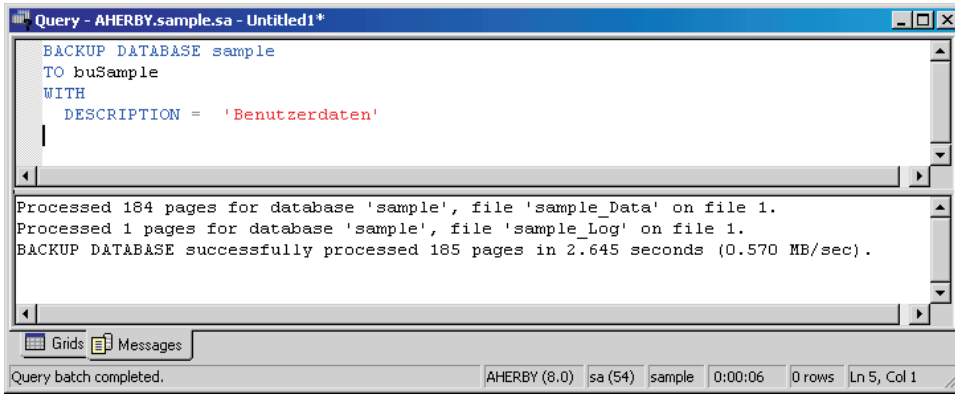


Abbildung 9.26: TSQL-Anweisung für das Backup im Query Analyzer

In der oberen Fensterhälfte steht die SQL-Anweisung, die ausgeführt wurde. Im unteren Bereich stehen einige Systemangaben, wie Datenseiten, Sicherungszeit und beteiligte Datenbankdateien.

PostgreSQL

Soll eine Datenbank in PostgreSQL gesichert werden kann dies mit Hilfsprogrammen geschehen. Für eine Sicherung im SQL-DUMP-Modus könnte das Programm *pg_dump* eingesetzt werden. Bei *pg_dump* handelt es sich um ein reguläres Client-Programm von PostgreSQL, es kann also von jedem Computer ausgeführt werden, der Zugriff auf die Datenbank hat. Voraussetzung ist Leseberechtigung für alle Datenbankobjekte. Vorteilhaft wäre also, sich als *Superuser* anzumelden.

Ausführungszeile:

```
pg_dump dbname > outfile
```

9.8 Wiederherstellung bei Datenverlusten

Weiter oben haben wir festgestellt, dass die Datenbank durchaus für Unfälle anfällig ist. Sie haben die zuvor vorgestellten Methoden genutzt, um ihre Datenbank zu sichern? Dann stellen Datenverluste, ob sie nun durch unbeabsichtigtes Löschen oder Kopflandung der Festplatte zustande gekommen sind, keinen Grund zur großen Trauer dar (wenn wir von dem Arbeitsaufwand einmal absehen). Es gilt zunächst das Ausmaß des Schadens festzustellen. Nicht immer muss eine Datenbank vollständig wiederhergestellt werden, wenn Fehler aufgetreten sind.

Fehlerdiagnose	Mögliche Fehlerursache	Fehlerbehebung
Datenbankanwendung kann nicht gestartet werden.	Authentifizierungsdaten sind falsch oder auf sie kann nicht zugegriffen werden. Anwendungsdateien sind beschädigt oder gelöscht. Nach der Einspielung eines Updates kommt es zu Versionskonflikten. Keine Verbindung zur Datenbank möglich. Programmbibliotheken (DLLs) wurden durch Installation eines anderen Programms überschrieben.	Kenndaten überprüfen. Vollständigkeit der Anwendungsdateien überprüfen. Datenbankverbindung mit anderem Programm testen. Version der Programmbibliotheken überprüfen. Ggf. Neuinstallation der Anwendung.
Funktionalität der Anwendung ist nur noch eingeschränkt gegeben.	Versionskonflikte. Teile der Datenbank fehlen oder sind beschädigt. Kein Zugriff auf Systemdateien. Programmbibliotheken (DLLs) wurden durch Installation eines anderen Programms überschrieben	Konfiguration der Anwendung überprüfen. Anwendung an einem anderen Clientrechner testen. DBMS auf Funktionalität überprüfen. Berechtigungen überprüfen und neu zuweisen. Ggf. Neuinstallation der Anwendung oder des DBMS.

Tabelle 9.3: Analyse bei Programm- und Datenbankfehlern

Fehlerdiagnose	Mögliche Fehlerursache	Fehlerbehebung
Daten werden in Berichten oder Übersichtstabellen nicht mehr angezeigt.	Es wurde ein falscher Bereich ausgewählt.	Vorgang mit neuer Selektion durchführen.
	Daten wurden gelöscht.	Abfragen direkt am DBMS ausführen.
	Abfragen sind fehlerhaft.	Konfiguration des Berichtes oder der Steuerelemente testen.
	Es sind Filter gesetzt.	Gelöschte Daten aus einer anderen Datenquelle importieren.
		Datenbank wiederherstellen.
Daten in Berichten oder Übersichtstabellen sind unvollständig.	Falsche Ansicht.	SQL-Anweisung direkt am DBMS ausführen.
	Falsche Selektion.	Ggf. Berichte oder Tabellen korrigieren.
	Daten wurden gelöscht.	Gelöschte Daten aus einer anderen Datenquelle importieren.
		Datenbank wiederherstellen.
Anwendung kann nicht mehr mit der Datenbank verbunden werden.	Falsche Authentifikationsdaten.	Kenndaten überprüfen.
	Schnittstelle ist falsch konfiguriert.	Test auf einem anderen Client durchführen.
	Fehlerhafte Netzwerkeinstellungen.	Netzwerkverbindung überprüfen und u.U. neu konfigurieren.
	Treiberdateien wurden gelöscht oder überschrieben.	Schnittstelle neu aufbauen.
		Treiber erneut installieren.
	Datenbank ist beschädigt oder gelöscht.	Datenbank wiederherstellen.
Das DBMS wird nicht mehr gestartet.	Benötigte Dienste sind noch nicht gestartet.	Alle notwendigen Dienste des Betriebssystems und des DBMS starten.
	Systemdateien sind gesperrt, beschädigt oder gelöscht.	Neuinstallation des DBMS.
Es ist nicht möglich, sich direkt am DBMS anzumelden.	Falsche Authentifikationsdaten.	Kenndaten checken.
	Es sind nicht alle Dienste gestartet.	Dienste starten.
Die Datenbank existiert (scheinbar) nicht mehr.	Falsche Ansicht.	Andere Ansicht wählen.
	Teile oder die gesamte Datenbank wurde gelöscht.	Datenbank wiederherstellen.

Tabelle 9.3: Analyse bei Programm- und Datenbankfehlern (Forts.)

Fehlerdiagnose	Mögliche Fehlerursache	Fehlerbehebung
Datenbankobjekte fehlen.	Fehlerhafte Transaktionsausführung.	Transaktionsprotokoll überprüfen.
	Objekte fehlen im Installationsskript.	Installationsskript korrigieren.
	Es wurden im Programm gleiche Objektnamen vergeben.	Datenbankobjekte erneuern und mit Daten füllen.
		Abhängige Programmodule testen.
Datenbankobjekte sind fehlerhaft.	Fehler in der Anwendung oder im Installationsskript.	Test und Korrektur in fehlerhafter Programmeinheit.
	Verletzung der Integritätsbedingungen.	Maßnahmen zur Integritätserhaltung einführen.
		Skripte überarbeiten und erneut ausführen.

Tabelle 9.3: Analyse bei Programm- und Datenbankfehlern (Forts.)

Der Tabelle können wir entnehmen, dass manchmal mehr Aufwand notwendig ist, um Datenverlust zu bereinigen. Im schlimmsten Fall muss das DBMS neu installiert und erst dann die Daten wiederhergestellt werden. In vielen Fällen gibt es aber völlig andere Ursachen, wenn die Anwendungen nicht mehr funktionieren. Trotzdem wird dann unnötigerweise oft das komplette System neu aufgebaut. Natürlich gibt es auch Fälle, wie die eingangs genannten, bei denen eine Wiederherstellung der Datenbank unumgänglich ist.

Darüber hinaus ist das Sichern und Wiederherstellen von Datenbanken für andere Zwecke sinnvoll, wie z. B. beim Kopieren einer Datenbank vom einem Server auf einen anderen Server. Durch das Sichern einer Datenbank von einem Computer und das Wiederherstellen der Datenbank auf einem anderen Computer kann schnell und problemlos eine Kopie der Datenbank erstellt werden.

MS SQL Server

Mit dem Enterprise Manager ist die Wiederherstellung einer Datenbank recht unkompliziert. Einige wenige Schritte sind erforderlich und Sie können wieder auf ihre verlorenen Daten zugreifen.

1. Im Menü *Werkzeuge* wählen Sie DATENBANK WIEDERHERSTELLEN.
2. Im geöffneten Dialogfeld DATENBANK WIEDERHERSTELLEN wählen Sie die Datenbank, die wiederhergestellt werden soll.
3. Sicherungskopie aus der Auswahlliste auswählen.
4. Getroffene Auswahl mit OK bestätigen.

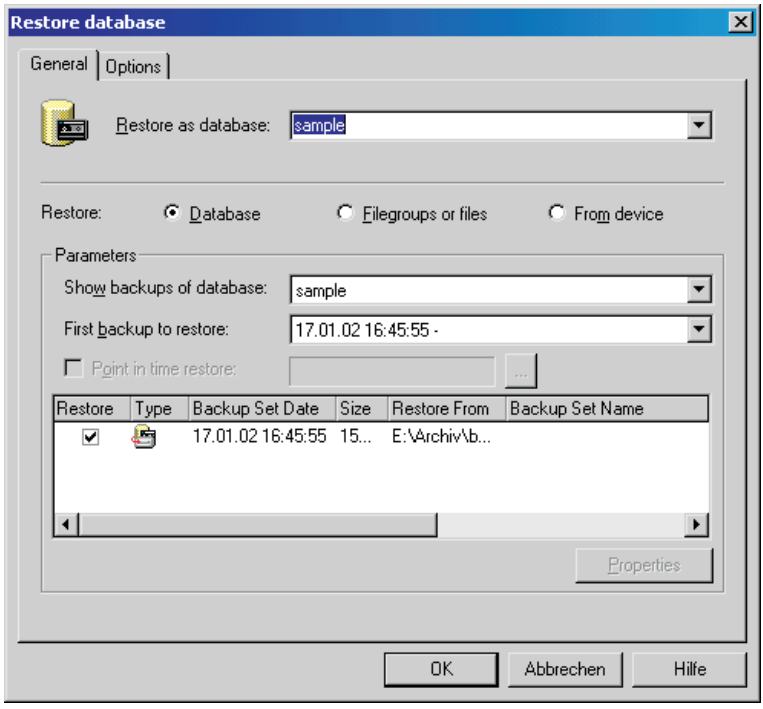


Abbildung 9.27: Wiederherstellung der DB beim SQL Server

Weitere Einstellungen, wie den Wiederherstellungsstatus, können Sie in der Registerkarte OPTIONEN vornehmen. Für die Wiederherstellung steht beim SQL Server kein Assistent zur Verfügung. Mit der `RESTORE DATABASE` gibt es jedoch wieder eine SQL-Anweisung, mit der die Datenbank wiederhergestellt werden kann. Manchmal ist es nötig, zusätzlich die *master*-Datenbank mit den Systemdaten neu zu erstellen:

1. Beenden des SQL Server.
2. Öffnen Sie die MS-DOS Eingabeaufforderung.
3. Starten Sie das Dienstprogramm REBUILD MASTER, indem Sie REBUILDM.EXE eingeben.
4. Wählen Sie Quell- und Zielverzeichnis, wie es in Bild 9.28 dargestellt ist (die master-Datenbank sowie andere Systemdatenbanken finden Sie auf der SQL Server-CD im Verzeichnis \DATA).
5. Wenn Sie auf EINSTELLUNGEN klicken, können Sie weitere Einstellungen zum Zeichensatz und Sortierreihenfolge vornehmen.
6. Aktivieren Sie die Befehlsschaltfläche ERNEUT ERSTELLEN, um den Vorgang zu starten.

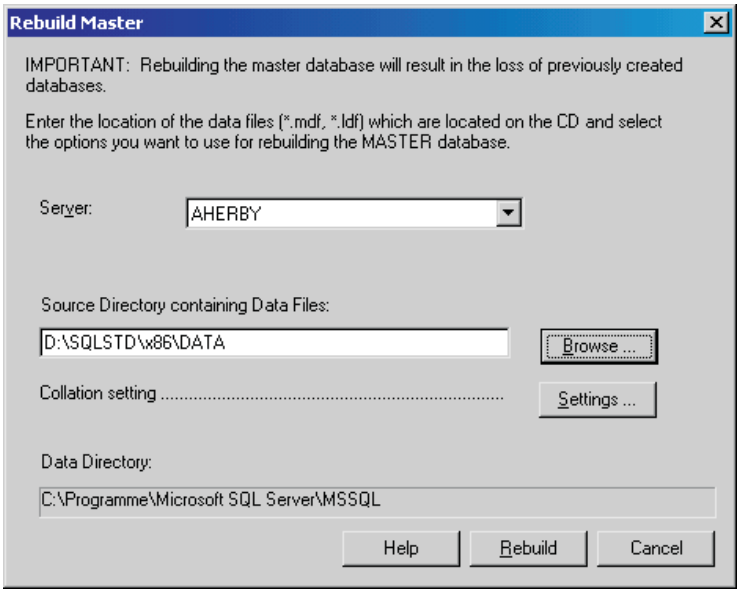


Abbildung 9.28: Der Rebuild Master

PostgreSQL

Auch bei PostgreSQL kann die zuvor erstellte Datenbanksicherung mit einigen wenigen Schritten wiederhergestellt werden. Wurde ein Backup mit *pg_dump* erstellt, so ist die Textdatei mit den SQL-Anweisungen Ausgangspunkt für die Wiederherstellung. Diese Datei kann mit dem Programm *psql* ausgelesen werden. Zunächst muss die Datenbank angelegt sein.

1. Erstellen der Datenbank

```
createdb -t template0 sample
```

2. Wiederherstellen aus Sicherungsdatei

```
psql sample < infile
```

Haben die Datenbankobjekte verschiedene Besitzer, so wird bei der Wiederherstellung nacheinander eine Verbindung der verschiedenen Benutzer aufgebaut und die Objekte erstellt. Für den Wiederherstellungsprozess bedeutet dies, dass die Benutzer alle existieren müssen und sie die Berechtigung haben müssen, sich als jeder von ihnen anzumelden. Bei der Wiederherstellung ist es ratsam, die Einstellungen zur Überprüfung der Authentifikation zu deaktivieren.

Mit *pg_dump* und *psql* können auch Speicherauszüge von einem Server zu einem anderen portiert werden.

9.9 Bereinigung von Fragmentierung

Im Zusammenhang mit Datenbanken hat der Begriff *Fragmentierung* eine doppelte Bedeutung. Zum einen wird darunter das gewünschte Verteilen von Daten auf mehrere Partitionen verstanden, zum anderen sind Datenbankdateien und insbesondere Indizes von »Zerstückelung« der Daten an ihrem physikalischen Speicherort betroffen. In diesem Abschnitt soll es um die unerwünschte Fragmentierung und ihre Bereinigung gehen.

9.9.1 Wie entsteht Fragmentierung?

Fragmentierung entsteht beim Abspeichern der Dateien auf der Festplatte. Viele Dateien sind in ihrer Größe nicht fest, sondern werden im Laufe der Zeit vergrößert, verkleinert oder gelöscht. Damit diese Aktionen möglichst schnell ausgeführt werden, versucht das Betriebssystem die physikalische Speicherung der Datei so wenig wie möglich zu ändern, d.h. im Falle einer Vergrößerung der Datei werden zusätzliche freie Speicherplätze belegt. Diese liegen aber häufig nicht anschließend an den ursprünglichen Speicherplätze, so dass die *Dateifragmente* über die Festplatte verteilt werden. Je häufiger Änderungen an Dateien durchgeführt werden, desto größer ist die entstehende Fragmentierung.

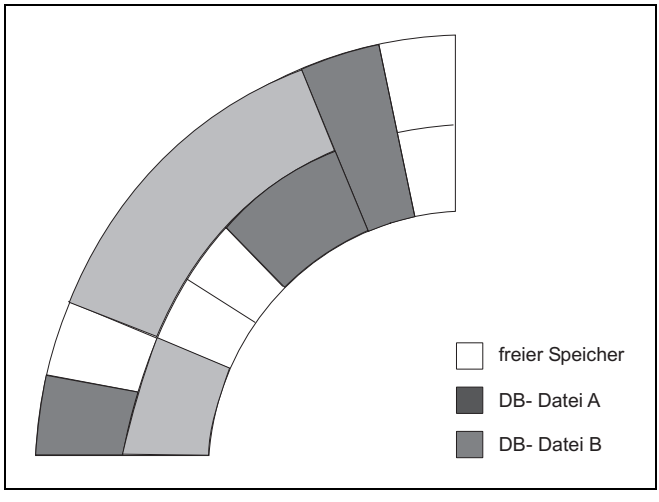


Abbildung 9.29: Fragmentierung auf der Festplatte

9.9.2 Fragmentierung in der Datenbank

Wenn Sie eine Datenbank mittels Dateien und Dateigruppen erstellen, müssen Sie eine Anfangsgröße für die Datei angeben. Diese Dateien werden beim Hinzufügen von Daten zu der Datenbank allmählich gefüllt. Sie müssen jedoch in Erwägung ziehen, ob und wie die Datenbank über den anfangs reservierten Speicherplatz hinaus vergrößert werden kann, wenn mehr Daten zur Datenbank hinzugefügt werden, als die Dateien aufnehmen können.

Standardmäßig werden Datendateien bei Bedarf so lange vergrößert, bis der Speicherplatz des Datenträgers verbraucht ist. Wenn Datenbankdateien nicht über die Anfangsgröße hinaus vergrößert werden sollen, muss dies bei Erstellung der Datenbank angegeben werden.

Alternativ dazu gibt es die Möglichkeit, Datendateien zu erstellen, die beim Füllen mit Daten automatisch vergrößert werden, jedoch nur bis zu einer zuvor festgelegten Maximalgröße. Auf diese Weise kann verhindert werden, dass der gesamte Speicherplatz einer Festplatte verbraucht wird.

Wird die automatische Vergrößerung von Dateien zugelassen, kann dies zur Fragmentierung der Dateien führen, wenn für eine große Anzahl an Dateien derselbe Datenträger verwendet wird. Aus diesem Grund wird empfohlen, Dateien und Dateigruppen auf so vielen unterschiedlichen verfügbaren lokalen physischen Datenträgern wie möglich zu erstellen. Verteilen Sie Objekte, die viel Speicherplatz beanspruchen, auf unterschiedliche Dateigruppen.

9.9.3 Fragmentierung bei Indizes

Indizes können ebenso störende *Fragmentierung* aufweisen. Werden in eine gegebene Tabelle beispielsweise neue Zeilen hinzugefügt, so sorgt das DBMS für die Anpassung der *Indexseiten*, vorausgesetzt die Tabelle enthält Indizes. Durch *Seitenteilung* können die vorgenommenen Änderungen an der Tabelle auch auf der Ebene der Indexseiten vollzogen werden und die entsprechenden Daten werden gespeichert. Durch den Teilungsvorgang nimmt die Größe des belegten Speicherplatzes für die Indexseiten zu und der Platz kann unter Umständen nicht mehr effektiv genutzt werden. Es wird also mehr Platz belegt als nötig und für das Durchsuchen der Tabelle kann dies mehr Lesevorgänge und damit eine erhöhte Abfragezeit bedeuten. Eine Fragmentierung der Indexseiten kann aber auch von Vorteil sein. Durch die beim Teilungsvorgang neu entstandene, nicht komplett ausgefüllte, Indexseite gibt es freien Platz, der beim Hinzufügen weiterer Zeilen in die Tabelle genutzt werden könnte. In diesem Fall wäre dann das aufwändige Aufteilen der Indexseiten nicht mehr nötig.

Eine Fragmentierung tritt auch dann auf, wenn die logische Reihenfolge der Seiten nicht der physischen entspricht bzw. die Blöcke, welche zu einer Tabelle gehören, nicht zusammenhängen. Diese Fragmentierungsart nennt man auch *externe Fragmentierung*.

9.9.4 Erkennen von Fragmentierung

Eine regelmäßige Analyse der Datenbankprozesse gehört zu den *Wartungsaufgaben* eines Datenbankadministrators. Die DBMS stellen von sich aus meist Werkzeuge in Form von Hilfsprogrammen zur Verfügung, mit denen die Arbeitsauslastung analysiert werden kann. Wird das System ohne offensichtliche Veränderungen an der Konfiguration bei Abfrageprozessen langsamer, dann könnte eine Ursache starke Fragmentierung sein und spätestens dann ist eine genauere Analyse erforderlich. Am Beispiel der Tabelle *kunde* und des DBMS MS SQL Server soll vorgeführt werden, wie eine Analyse aussehen könnte.

1. Der Query Analyzer wird gestartet.
2. Ausführen von DBCC SHOWCONTIG (siehe Abbildung 9.30).
3. Auswerten der Ergebnistabelle, die in Abbildung 9.30 zu sehen ist.

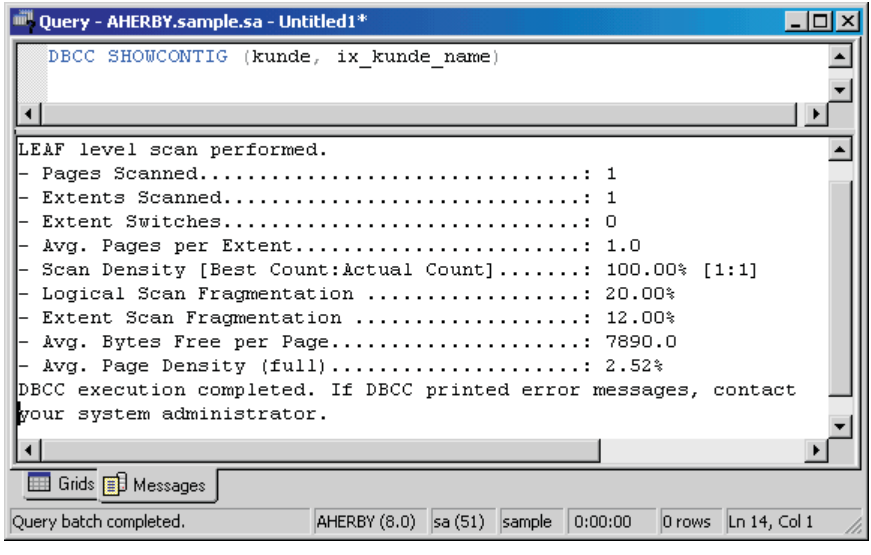


Abbildung 9.30: Analysedaten im Query Analyzer

Die Angaben zur *Blockscanfragmentierung* (Extent Scan Fragmentation), *Logische Scanfragmentierung* (Logical Scan Fragmentation) und *Scandichte* (Scan Density) sollten zur Beurteilung von Fragmentierung besonders berücksichtigt werden. Je niedriger der

Wert für Blockscanfragmentierung und logische Scanfragmentierung, desto weniger Fragmentierung liegt vor. Der Wert der Scandichte ergibt sich aus der Division von der optimalen Anzahl der Blockwechsel und der tatsächlichen Anzahl. Der Wert 100%, wie im Beispiel, ist optimal.

9.9.5 Defragmentierung

Mit den Systemwerkzeugen Ihres Betriebssystems oder Programmen von Drittanbietern können Sie die Dateifragmentierung Ihrer Festplatte bereinigen. Dabei werden die Dateien reorganisiert, so dass die Zugriffszeiten beim Laden minimiert werden. Die Daten können wieder effektiv genutzt werden und liegen wieder in zusammenhängenden Stücken auf der Festplatte.

Um die Fragmentierung aus einem Index zu entfernen, kann eine von den angegeben Methoden verwendet werden:

- Index mit `DROP INDEX` und `CREATE INDEX` neu erstellen;
- Alle Indizes einer Tabelle mit `DBCC DBREINDEX` neu anlegen;
- Defragmentierung ohne Neuerstellung mit `DBCC INDEXDEFRAG`.

Das Beispiel 9.7 zeigt, wie ein Index defragmentiert wird, ohne dass eine Neuerstellung erfolgt.

Beispiel 9.7:

```
DBCC INDEXDEFRAG (sample,kunde,ix_kunde_name)
```

Als Ergebnis wird bei korrekter Ausführung eine Ergebnismenge mit den gescannten und bearbeiteten Indexseiten angegeben. Bei umfangreichen Tabellen kann der Vorgang der *Defragmentierung* durchaus einige Minuten andauern.

9.10 Wartungspläne

Es gibt einige Routineaufgaben, zu denen auch Sicherung von Datenbank und Protokollen gehören, die der Administrator zu erledigen hat. Diese Aufgaben müssen nicht immer manuell durchgeführt werden, sondern können auch in einem Wartungsplan spezifiziert werden und dann vom DBMS zum gegebenen Zeitpunkt ausgeführt werden. Bei einigen DBMS steht die Funktionalität eines Wartungsplaners standardmäßig zur Verfügung, bei anderen muss dies durch Erstellung von Skripten erledigt werden. Eine Auflistung der Aufgaben, die Bestandteil eines Wartungsplans sein können, soll die Bandbreite der Möglichkeiten aufzeigen:

- Reorganisation der Daten- und Indexseiten,
- Freigeben von unbenutztem Speicher,
- Update-Statistik,
- Integritätscheck,
- Sicherung der Datenbank u. des Transaktionsprotokolls,
- Berichtgenerierung,
- Protokollierung von Datenbankaktivitäten.

Beim MS SQL Server läuft diese Planung von Wartungsaufgaben über den *SQL Server Agent*. Der Server Agent ist ein Dienst, der SQL-Anweisungen aus einer Tabelle liest und diese ausführt. Dadurch können Aufgaben in Form eines Wartungsplans definiert werden und zu definierten Zeitpunkten abgearbeitet werden. Ein Auftrag kann so definiert werden, dass er beim Starten des SQL Server-Agents, im Leerlauf (definierter Bereich der CPU-Auslastung), zu einem angegebenen Zeitpunkt und Datum, auf wiederkehrender Basis oder als Antwort auf eine Warnung gestartet wird.

Einen Wartungsplan kann man über den Datenbankwartungsplanungs-Assistenten halbwegs automatisiert erstellen lassen. In den einzelnen Dialogfelder werden Parameter zu den einzelnen Aufgaben abgefragt und abschließend wird vom DBMS der Wartungsplan erzeugt. Gemäß der angegebenen Zeitangaben werden die Aufgaben ausgeführt.

Im Enterprise Manager kann eine Aufgabe auch manuell angelegt werden, indem zumindest ein Aktionsschritt definiert werden muss.

1. Erweitern der Konsolenstruktur bis VERWALTUNG (Management) erscheint.
2. Erweitern Sie Verwaltung und dann SQL Server-Agent.
3. Klicken Sie mit der rechten Maustaste auf AUFTRÄGE (Jobs) und im Kontextmenü auf NEUER AUFTRAG.
4. Im geöffneten Dialogfeld (siehe Abbildung 9.31) die Registerkarte ALLGEMEIN (General) auswählen.
5. Geben Sie einen Namen im Namensfeld an.
6. Deaktivieren Sie das Kontrollkästchen AKTIVIERT, wenn der Auftrag nicht direkt nach der Erstellung aktiviert werden soll.
7. Machen Sie weitere Angaben zum Besitzer, Multi-Server-Betrieb, Beschreibung und Kategorie oder verwenden Sie die Standardeinstellungen.
8. Wechseln Sie zur Registerkarte AUFTRAGSSCHRITTE (Steps).

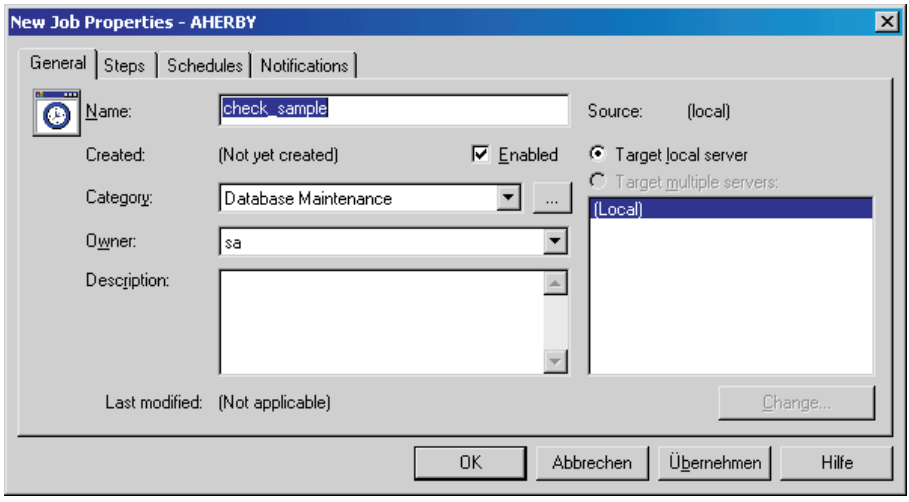


Abbildung 9.31: Der SQL Server-Agent

9. Klicken Sie auf die Befehlsschaltfläche NEU..., um einen Auftragsschritt zu definieren.

10. Geben Sie den Namen, Typ und die Datenbank an.

11. Laden Sie ein SQL-Skript oder schreiben Sie SQL-Anweisungen in das Kommandofeld.

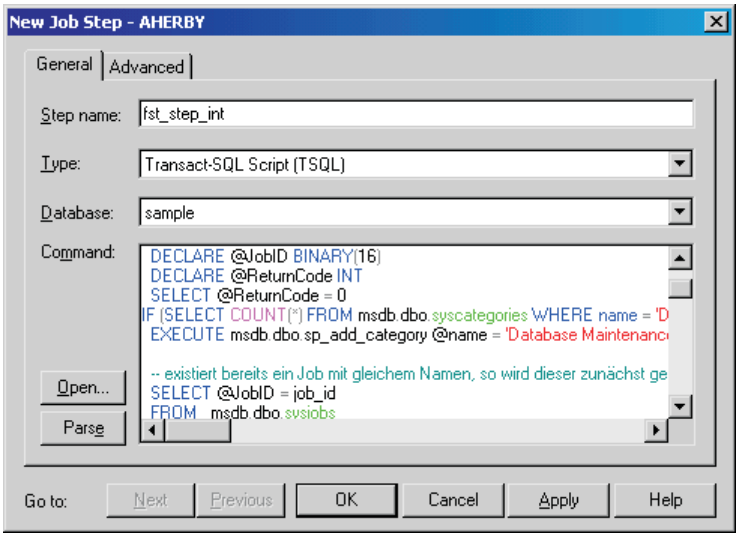


Abbildung 9.32: SQL-Anweisung für einen neuen Job

12. In den weiteren Registerkarten können Sie noch den Ausführungstermin planen und Nachrichten an Operatoren versenden.

13. Nach dem Klicken auf OK wird der Auftrag angelegt.

Aufträge können so geschrieben werden, dass sie auf der lokalen Instanz von SQL Server oder auf mehreren Servern ausgeführt werden. Sie können sich selbst ein SQL-Skript in einem Texteditor oder im Query Analyzer erstellen und dies dann in das Kommandofeld laden. Sie können auch das Skript INDEXCHECK.SQL von der Begleit-CD laden. Mit diesem Skript wird die Datenintegrität der Datenbank *sample* inklusive der Indizes überprüft.

9.11 Sicherheit im Internet

Wenn das Internet für den kommerziellen Bereich genutzt wird, ist die Sicherheit und der Schutz von Firmengeheimnissen ein wichtiges Thema. Mit dem Anschluss eines Rechnernetzes an das Internet entstehen für die im Netzwerk gespeicherten Daten eine Vielzahl von Gefahrenquellen. Allgemein kann das Sicherheitsproblem beim Anbinden an das Internet in drei Risikogruppen eingeteilt werden:

- Verwendete Protokolle zur Datenübertragung,
- Programme für Dienstabwicklung,
- Verwendete Systeme selbst.

Die im Internet verwendeten *Protokolle* haben keine sicheren Mechanismen zur Identifikation und Authentifizierung im Netz. Häufig werden auch Programme gestartet, die nicht notwendig sind und nur zu zusätzlichen Sicherheitslücken führen. Hierzu gehören z.B. Server-Programme auf einem Rechner, der nur zum Abrufen von Informationen vorgesehen ist. Voreingestellte, umfangreiche und schlecht dokumentierte Konfigurationsdateien führen ebenfalls häufig zu Gefährdungen. Im Internet werden alle Informationen grundsätzlich offen übertragen, so dass sie von jedem, der Zugang zu einem benutzten Netz hat, mitgelesen werden können. Dies gilt insbesondere für Passwörter und die elektronische Post (E-Mail), die deshalb mit einer nicht unterschriebenen Postkarte verglichen werden kann. Außerdem werden alle Daten im Internet ohne einen Schutz gegen Veränderungen übertragen, so dass jeder, der Zugang zu einem beteiligten Netz hat, die Daten manipulieren kann. Dieses sind nur einige Angriffsflächen, die man bei einer Internetanbindung berücksichtigen muss. Auch für das Arbeiten im Internet soll im Folgenden ein Maßnahmenkatalog vorgestellt werden, welcher durch Anwendung die Sicherheit erhöhen wird:

- Beschaffung von Informationen über alle Prozesse, die zu einer Internetverbindung führen können.
- Auswahl der wirklich benötigten Dienste.
- Kenntnisnahme der beteiligten Protokolldienste und ihr Speicherort.
- Auf einem Rechner, der ohne weiteren Schutz mit einem unsicheren Netz verbunden ist, sollten besonders gefährdete Dienste wie NFS oder NIS deaktiviert werden.
- Abschalten und evtl. Löschen der ungenutzten Programme.
- Entfernen der ungenutzten Dateien aus Startup-Dateien und Anpassen von Konfigurationsdateien.
- Abschalten der automatischen Ausführung aktiver Inhalte im verwendeten www-Browser.
- Netzwerkanalyse um Konzeptionsfehler auszuschließen.
- Einsatz aller Sicherheitsmaßnahmen des Betriebssystems (Rechtevergabe, Backups, Protokollmechanismen etc.).
- Einsatz von Programmen, die Integritätsverletzungen an Programmen und Dateien feststellen können.
- Verschlüsselung auf der Anwendungsschicht oder Betriebssystemebene bei der Übertragung der Daten.
- Für größere Computernetze sollte ein zentraler Übergang zum Internet geschaffen werden (Einsatz von Firewalls).

9.11.1 MS SQL Server und Internet Information Server

Microsoft hat die Kritik und die Sorgen der Kunden um die Sicherheit ihrer Systeme mittlerweile ernst genommen und bietet für die Überprüfung des Internet Information Server (IIS) und SQL-Datenbanken nach Sicherheitslücken Hilfswerkzeuge. Diese Administrations-Hilfswerkzeuge können von der Microsoft-Seite im Internet heruntergeladen werden (die Adresse finden Sie im Anhang). Der IIS hat in der Vergangenheit den Hackern oftmals das Leben recht einfach gemacht. Sein Einsatz und seine Funktionalität werden Sie im Kapitel 11 kennen lernen. Bei der Konfiguration müssen Sie etwas Handarbeit verrichten, damit Sie die Angriffsfläche für Hacker verringern.

Hier einige Tips für die IIS-Konfiguration:

- Informieren Sie sich über aktuelle Sicherheitsupdates und spielen Sie diese baldmöglichst ein.
- Machen Sie von der Protokollierungsmöglichkeit Gebrauch.

- Verwenden Sie, falls möglich, ein englisches Betriebssystem, weil die Sicherheitsupdates hierfür schneller verfügbar sind.
- Passen Sie nach der Installation des IIS die Zugriffsrechte für das Dateisystem an.
- Deaktivieren Sie nicht benutzte Benutzerkonten.
- Deaktivieren Sie unnötige Standardskripte.
- Lagern Sie das www-Verzeichnis auf einer gesonderten Partition aus.
- Lassen Sie IIS unter einem separaten Prozess laufen (Einstellung unter Eigenschaften im Service Manager).
- Deaktivieren Sie alle nicht benötigten Dienste (z.B. Fax Service, Messenger, Telnet etc.).

10 Berichterstellung

Auch wenn elektronische Medien dem Papier als Informationsträger den Rang ablaufen wollen, so hat das Erstellen eines Berichtes doch einen wichtigen Platz im Gesamtkonzept der Datenbankanwendung. Erstens ist der Verzicht auf Papier noch lange nicht abzusehen und zweitens sind auch für die elektronischen Medien Berichte unverzichtbar. Es ist kaum denkbar, dass der Kollege aus der Marketingabteilung für seinen Vortrag einfach eine Abbildung der Tabellen akzeptiert, wenn er den Kunden Produktinformationen aus der Datenbank vorführen möchte. Auch Ihr Chef wird wenig begeistert sein, wenn er statt der Liste mit den Fehlstunden einen recht kryptischen Ausdruck mit Mitarbeiter- und Leistungsdaten bekommt, bei dem er sich die Relationen selbst zusammenbasteln muß. Kurz und gut, immer wenn Zwischenstände, Ergebnisse oder Analysen benötigt werden, kommt man nicht an einer Berichterstellung vorbei. Außerdem werden Serienbriefe, Rechnungen, Stücklisten etc. mit Werten aus der Datenbank automatisch generiert.

10.1 Allgemeine Richtlinien – das Auge isst mit

Die richtige Planung ist wesentliches Element bei der Erstellung eines Berichtes. Dabei sollte das Ziel klar beschrieben und verfolgt werden. Als Ziel lässt sich in erster Linie die Zufriedenheit des Auftraggebers spezifizieren. Arbeitsergebnisse sollen in angemessener Form präsentiert werden. In Ergebnislisten zu »wühlen« ist oft eine sehr ernüchternde Arbeit, die durch einen unstrukturierten Bericht zusätzlich erschwert wird. Machen Sie sich und Ihren Mitmenschen das Leben nicht unnötig schwer. Bemühen Sie sich, die folgenden Richtlinien zu befolgen:

Layout

1. Zügeln Sie Ihre Kreativität und halten Sie sich an Standards und Vorgaben.
2. Entscheiden Sie sich in Abhängigkeit von der Datenmenge für das richtige Berichtsformat.
3. Arbeiten Sie nicht mit allen Schriftarten, die Ihnen zur Verfügung stehen.

4. Versehen Sie jede Seite des Berichtes mit Datum, Seitennummer und Gesamtzahl der Seiten.
5. Denken Sie an einheitliche Ausrichtung der verschiedenen Elemente im Bericht.
6. Überlegen Sie, welche Daten von besonderer Wichtigkeit sind und formatieren Sie diese entsprechend.
7. Vermeiden Sie Abschneidungen bei den eingefügten Daten, indem Sie die Feldbreite ausreichend groß definieren.
8. Achten Sie auf die Blattaufteilung, Ränder und Zeilenabstände.
9. Gehen Sie mit dem Einsatz von Farben sparsam um.
10. Absatzgliederung wählen (Geviert-Einzug, Laisser-faire-Stil oder Hängender Einzug).
11. Gestalten Sie Kopf- und Fußzeilen (Logo der Firma, Versionsidentifikation, Seitenzahl etc.).
12. Zu viele Schriftgrößen erzeugen Verwirrung beim Betrachter.

Ausführungsprozess

1. Vermeiden Sie Datenredundanz in den Berichten.
2. Denken Sie an die richtige Auswertung der NULL-Werte.
3. Übergeben Sie dem Reportgenerator nur die Daten, die er für den Bericht benötigt.
4. Beachten Sie beim Entwurf und beim Test des Berichtes, dass er unter Umständen auf verschiedenen Druckern mit unterschiedlichen Treibern ausgegeben wird. Verwenden Sie Standards bei den Schriftarten.
5. Verwenden Sie im Bericht nicht zu viele Programmanweisungen, um z.B. Zeilen zu unterdrücken oder Werte umzurechnen. Wenn nötig, verwenden Sie Standardfunktionen und übergeben Sie dem Bericht die zuvor umgerechneten Werte.
6. Halten Sie die Komplexität der SQL-Abfragen so gering wie möglich.
7. Denken Sie an Exportmöglichkeiten in andere Dateiformate. Testen Sie auch dort Funktionalität und Format.

10.2 Werkzeuge für die Berichterstellung

Es gibt DBMS, bei denen ein Reportgenerator zum Gesamtpaket gehört. Dies ist aber absolut nicht Standard. Im Übrigen ist es auch sinnvoll, diese Funktionalität eher den Entwicklungswerzeugen zuzuordnen, mit denen die Anwendung erstellt wird. Doch auch die Entwicklungsumgebungen können nicht immer mit Werkzeugen für die Berichterstellung glänzen, deshalb ist man bisweilen auf Produkte von Drittanbietern angewiesen.

10.2.1 Crystal Report

Der Marktführer bei den Reportgeneratoren ist die Firma *Seagate*. Crystal Report von Seagate ist mittlerweile für viele Entwickler ein unverzichtbares Werkzeuge geworden. Versionen mit eingeschränkter Funktionalität findet man bereits auch in Produkten anderer Hersteller (MS Visual Studio oder Netscapes Webserver). Mit dem Crystal Report Designer lassen sich Berichte sehr komfortabel in Ihre Anwendung einbinden. Mit Assistenten und Berichtmanagern lassen sich die Daten auswählen und das Erscheinungsbild des Berichtes den individuellen Bedürfnissen entsprechend anpassen.

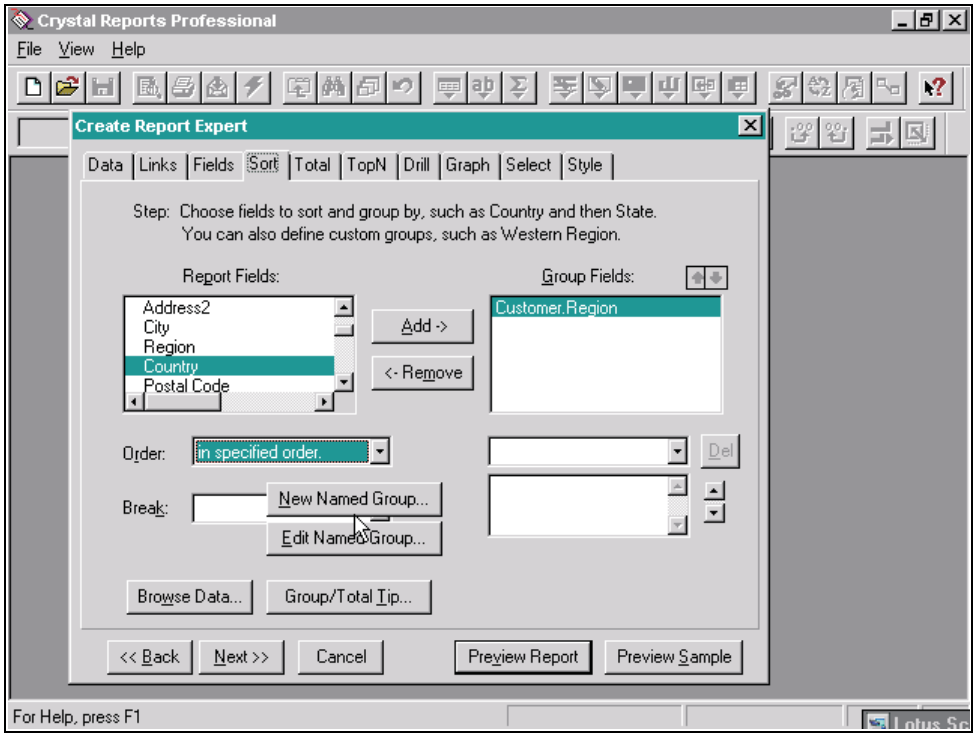


Abbildung 10.1: Benutzeroberfläche von Crystal Report

Datenbankkenntnisse sind trotz der Hilfsfunktionen unbedingt erforderlich. Die notwendigen Schritte zum gelungenen Bericht werden in einem Fenster mit Registerelementen übersichtlich angezeigt. Sobald der Benutzer alle wichtigen Daten ausgewählt hat, erzeugt der Assistent den gewünschten Report, dessen visuelles Erscheinungsbild sich noch optimieren lässt. Von Etiketten, über Serienbriefe, bis hin zu multidimensionalen Auswertungen von OLAP-Daten, ist die Bandbreite der gebotenen Berichtsformate sehr groß. Spezielle Informationen wie Druckdatum oder die Anzahl der Seiten lassen sich einfach in den Bericht einbinden. Die Umrechnung von Datenwerten lässt sich mit einem integrierten Formeleditor erledigen.

Zahlreiche Operatoren stehen für die Verfeinerung des Reports zur Verfügung. Arithmetische und logische Operatoren sind genauso vorhanden, wie Methoden für die Stringbearbeitung oder Datumsfunktionen.

Aus dem Diagramm-Manager kann aus zwölf vorgefertigte Diagrammen gewählt werden. Selbstverständlich können Sie Diagramme auch nach Ihrem eigenen Geschmack entwerfen. Wichtig ist dabei die Bildung von Gruppen- und Summenfeldern, damit das Programm die Daten in grafische Darstellungen umwandeln kann. Die Achsen und Diagramme können mit Texten und vielfältigen Optionen versehen werden. Mit dem neuen Map-Experten kann der Benutzer geografische Details an Daten anhängen, um beispielsweise Umsätze nach Ländern geordnet anzuzeigen und auszuwerten. (siehe Abbildung 10.2)

Mit der Einbindung von Crystal Query als Java-basiertem Programm ist es jetzt möglich, aus einem Browser über die MS JVM (Java Virtual Machine) einen Report anzuzeigen und mit Daten zu füllen. Der Leistungsumfang ist gegenüber dem Reportgenerator jedoch ebenso wie die Layoutbeeinflussung deutlich eingeschränkt. Neue Reports kann der Anwender ebenfalls über den Browser erzeugen. Die Qualität bleibt hierbei jedoch manchmal auf der Strecke.

Für den Einsteiger erscheint die Handhabung von Crystal Report vielleicht etwas ungewohnt. Viele Funktionen sind auf den ersten Blick nicht durchschaubar. Schließlich besticht das Produkt jedoch durch eine Vielzahl an Optionen.

Auf der Begleit-CD befindet sich eine Demoversion von Crystal Report, die Ihnen einen Eindruck der Software vermitteln kann.

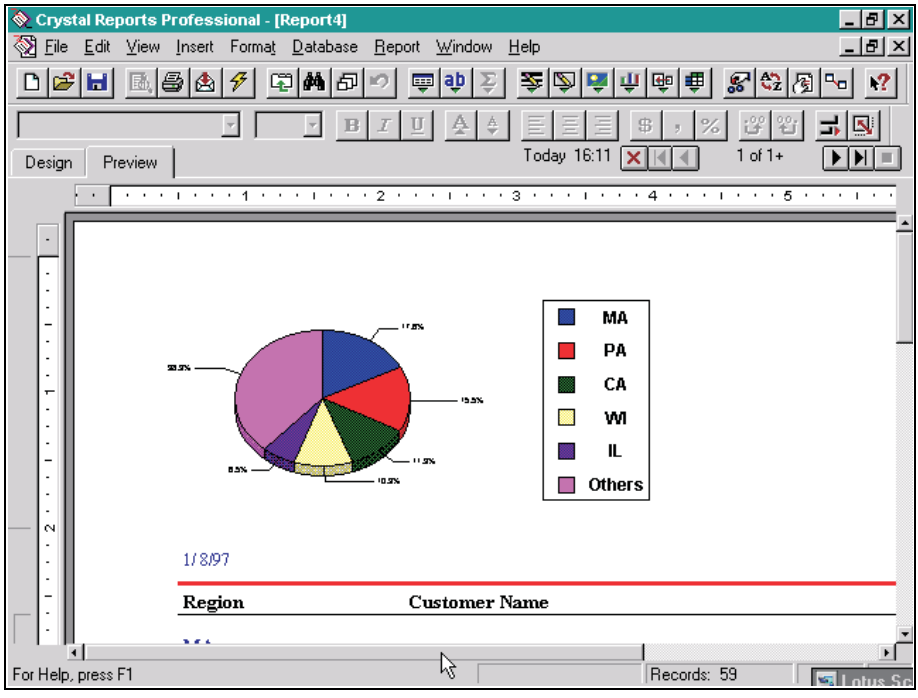


Abbildung 10.2: Diagramm in Crystal Report

10.2.2 ReportTool

ReportTool ist ein ausgereiftes Werkzeug zur Erstellung von Berichten in Publishing Qualität. Beliebige Daten werden automatisch aufbereitet, am Bildschirm dargestellt, ausgedruckt oder im Hintergrund in Dateien umgeleitet. Die Definition eines Berichtlayouts wird in einem grafischen Editor und mit einer einfachen Scripting-Sprache erstellt. Dank der umfangreichen Funktionalität und Flexibilität bietet ReportTool einfaches Database Publishing und Online Reporting.

Mit ReportTool werden Daten als korrekte und übersichtliche Informationen präsentiert, die als Basis zur Meinungsbildung, Entscheidungsfindung und Dokumentation dienen; zum Beispiel in Form von Arbeitsrapporten, Bestellungen, Inventarlisten, Rechnungen, Verkaufsübersichten oder Auswertungen. Die Endanwender werden entlastet und in ihrer eigentlichen Funktion unterstützt.

Bei den Berichten ist die stabile organisatorische Einbindung der Erstellung wichtig, damit die Korrektheit und übersichtliche Vergleichbarkeit über Perioden hinweg gegeben ist.

ReportTool wurde in C entwickelt und läuft unter Unix (Sun Solaris, HP-UX, AIX, Linux).

10.2.3 MS Access

MS Access verfügt standardmäßig über ein Berichtsmodul, mit dem Sie Ihre Daten formal aufbereiten können. Berichte sind bei MS Access ebenso wie Tabellen oder Formulare Bestandteile der Datenbank und sie werden gemeinsam mit diesen in einer Datei abgespeichert. Sie können einen Bericht manuell erstellen oder vom Berichts-Assistenten erstellen lassen. Das Erstellen von Berichten mit Hilfe eines Berichts-Assistenten ist schneller, da dieser alle grundlegenden Aufgaben für Sie übernimmt. Wenn Sie einen Berichts-Assistenten verwenden, fordert dieser Sie auf, Informationen einzugeben und erstellt auf der Basis dieser Informationen einen Bericht. Anschließend können Sie zur Entwurfsansicht wechseln, um den Bericht an Ihre Erfordernisse anzupassen.

Die Daten können im Bericht auch in Form von Diagrammen (Säulendiagramm, Balkendiagramm, Flächendiagramm etc.) ausgewiesen werden.

Der Bericht kann in mehrere Bereiche aufgeteilt werden. Neben dem *Detailbereich* für Daten, können Kopf- und Fußzeilen auf der Ebene der Seite und auf Ebene des Formulars eingefügt werden. Optional haben Sie auch die Möglichkeit mehrere Unterberichte in einem Hauptbericht anzuordnen. Ein *Unterbericht* ist ein Bericht, der in einen anderen Bericht eingefügt wird. So können Sie auf der Basis von unterschiedlichen Tabellen oder Abfragen Daten in einem Bericht zusammenfassen.

Sie können Werte im Bericht mittels verwendeter Funktionen umformen oder umrechnen. Dazu steht Ihnen ein Formeleditor zur Verfügung. Weiterhin können Sie die ganze Funktionalität von VBA nutzen, um auf der Programmzeilenebene Formartierungsaufgaben zu erledigen oder andere Aktionen zu starten.

Der fertige Bericht kann in Fremdformate exportiert werden. Oftmals besteht der Bedarf, den Bericht und nur den Bericht elektronisch zu versenden. Sie können in Access den Bericht ins HTML-, RTF- oder Excel-Format exportieren. Bei dem Export-Vorgang wird jedoch Information über das Layout teilweise verloren gehen. Eine interessante Alternative bietet hier das Abspeichern des Berichtes als *Bericht-Snapshot*. Ein Bericht-Snapshot ist eine Datei, die Kopien jeder Seite eines Berichtes enthält. Die für den Bericht notwendigen Daten werden als Speicherauszug (Snapshot) dem Bericht-Snapshot beigefügt und sind Bestandteil der Berichtsdatei mit der Namenserweiterung SNP. Der Bericht kann anschließend verschickt werden (z.B. per E-Mail) und mit Access oder einem *Snapshot Viewer* geöffnet werden. Der Snapshot Viewer ist ein Programm, welches man zur Anzeige und zum Ausdrucken von Bericht-Snapshots verwenden kann. Dafür ist es nicht erforderlich, über die Datenbank oder überhaupt eine Access-Lizenz zu verfügen. Der Snapshot Viewer lässt sich als Steuerelement in eine Webseite einbetten. So ist es auch möglich, den Bericht im Internet anzuzeigen.

10.3 Bericht für die Beispieldatenbank

Mit dem Berichtsmodul aus Access soll in diesem Abschnitt vorgeführt werden, wie die Daten aus der Beispieldatenbank in Form eines Berichtes ausgewertet werden. Neben der Grundüberlegung, welche Daten in welcher Form ausgegeben werden müssen, sind folgende Schritte erforderlich:

- Daten für den Bericht vorbereiten,
- Bericht mit Datenquelle verbinden,
- Felder anordnen,
- Layout bearbeiten,
- Überprüfen der Werte.

Bei den angegebenen Arbeitsschritten ist es zunächst unerheblich, welches Werkzeug Sie für die Berichtserstellung verwenden. Die Handhabung der Werkzeuge ist verschieden. Die grundsätzliche Methodik ist jedoch dieselbe.

10.3.1 Daten für den Bericht vorbereiten

Mit der Anforderung, die für den Bericht gegeben ist, steht auch fest, welche Felder bzw. welche Informationen aus der Datenbank im Bericht enthalten sein sollen. Eine sorgfältige Vorbereitung kann sich später zugunsten von schnellen Ausführungszeiten bezahlt machen. Übergeben Sie nur die Felder an den Bericht, die auch direkt oder in umgewandelter Form im Bericht erscheinen. Je mehr Felder Sie übergeben, desto komplexer wird die Berichterstellung. Außerdem werden die nicht benötigten Felder unnötigerweise bei jedem Schritt, von der Abfrage bis zum Generieren des Berichtes, mitgeschleppt und dadurch Ressourcen gebunden.

Versuchen Sie anhand der Tabellenstruktur aus Ihrem Entwurf festzustellen, welche Tabellen die Basis für den Bericht bilden. Liefert eine einzige Tabelle sämtliche Informationen, so ist die Zusammenstellung der Daten recht einfach. Normalerweise werden jedoch Informationen aus mehreren Tabellen benötigt. Die Tabellen müssen also zunächst in einer Sicht (bzw. einer Abfrage in Access) zusammengestellt werden. Eine andere Möglichkeit ist das Extrahieren der Daten über eine Auswahlabfrage in eine temporäre Tabelle. Mit `SELECT...INTO` werden die Felder aus den verschieden Tabellen ausgewählt und an eine oder mehrere temporäre Tabellen übergeben. Der Berichtsgenerator öffnet dann die temporäre Tabelle und liest die benötigten Daten aus. Auch hier ist es ratsam, nicht zu viele Tabellen über einer SQL-Anweisung zu verknüpfen. Stellen Sie ggf. die Daten mit mehreren SELECT-Anweisungen zusammen, um die Laufzeit zu verringern.

Es soll ein Bericht mit den Auftragsdaten je Kunde in der Beispieldatenbank erstellt werden. Dafür muß in Access eine Abfrage definiert werden, welche die Daten aus den Tabellen *kunde* und *auftrag* zusammenfasst. Erstellen Sie die Abfrage, indem Sie die folgenden Punkte ausführen.

1. Öffnen Sie die Beispieldatenbank in MS Access.
2. Wählen Sie als Datenbankobjekt ABFRAGEN.
3. Klicken Sie auf die Befehlsschaltfläche NEU.
4. Im geöffneten Dialogfeld wählen Sie den Auswahlabfrage-Assistenten.
5. Suchen Sie die Felder aus der Auswahlliste (siehe Abbildung 10.3).

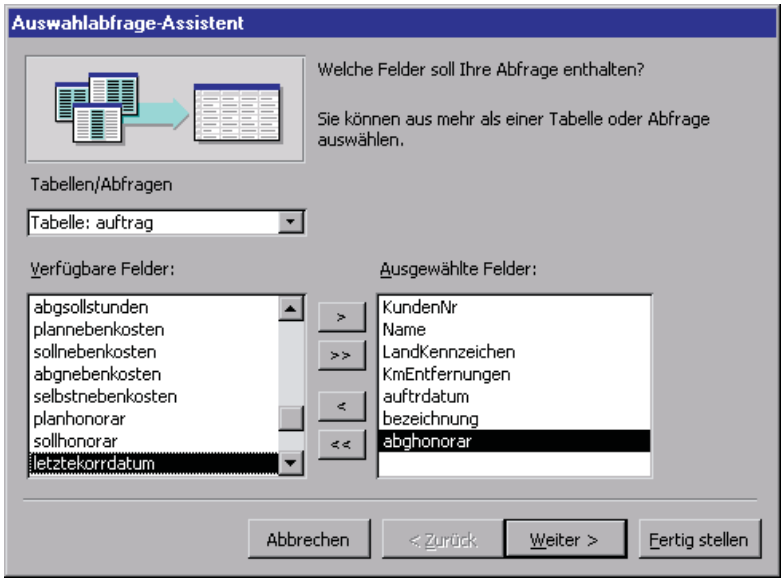


Abbildung 10.3: Der Auswahlabfrage-Assistent in MS Access

1. Wählen Sie im nächsten Dialogfeld die DETAILABFRAGE.
2. Vergeben Sie einen Namen für die Abfrage.
3. Klicken Sie auf FERTIGSTELLEN, um die Abfrage erstellen zu lassen.

Sie können die Abfrage nun testen. Die bisher definierten Abfragen, stehen im rechten Fenster, wenn Sie die Abfragen aufrufen. Öffnen Sie die soeben definierte Abfrage durch einen Doppelklick auf das Abfragesymbol. Die geöffnete Abfrage (siehe Abbildung 10.4) hat das gleiche Aussehen wie eine Tabelle in der Datenblattansicht.

qryAuftragKunde : Auswahlabfrage (lokal)

auftragnr	kundennr	auftrdatum	abghonorar	Name	LandKe	KmEntfernung
1	BEKL0001	03.04.2001	12.400,00 DM	Klaus	D	50
2	BEKL0001	02.02.2001	2.340,00 DM	Klaus	D	50
3	HHHE0001	05.12.2000	2.300,00 DM	Hemple	D	34
4	HHHE0001	01.01.2000	2.344,00 DM	Hemple	D	34
5	HHHE0001	04.06.2000	4.555,00 DM	Hemple	D	34
6	BEKL0001	23.09.2000	1.000,00 DM	Klaus	D	50
7	BEFH0002	15.12.2000	2.300,00 DM	Friedrich	D	

Datensatz: 1 von 7

Abbildung 10.4: Abfrageergebnis in MS Access

Ändern einer Abfrage im Entwurfsmodus

Die vorher definierte Abfrage soll nun für eine bestimmte Zeit spezifiziert werden. Es sollen nur die Aufträge für das Jahr 2000 berücksichtigt werden. In der WHERE-Klausel der SELECT-Abfrage muß deshalb nachträglich diese Einschränkung vorgenommen werden. Änderungen an eine Abfrage können in Access im Entwurfsmodus vorgenommen werden. Wählen Sie die Abfrage aus und klicken Sie auf ENTWURF. Die Abfrage wird dann, wie dies auch in Bild 10.5 zu sehen ist, im Entwurfsmodus geöffnet.

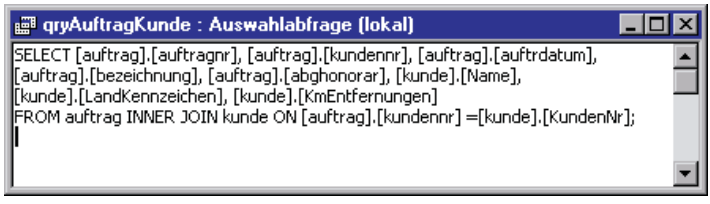


Abbildung 10.5: SQL-Code der Abfrage

Fügen Sie nun am Ende der Abfrage, noch vor dem Semikolon, die WHERE-Klausel an:

```
WHERE YEAR(auftrdatum) = '2000'
```

Die vollständige Anweisung definiert sich wie folgt:

```
SELECT auftrag.auftragnr, auftrag.kundennr, auftrag.auftrdatum,
auftrag.abghonorar, kunde.LandKennzeichen, kunde.KmEntfernungen, kunde.Name
FROM auftrag INNER JOIN kunde ON auftrag.kundennr = kunde.KundenNr
WHERE YEAR(auftrdatum) = '2000';
```

Die Funktion *YEAR* extrahiert das Datum aus dem Feld *auftrdatum*. Auf diese Weise werden nur die Aufträge ausgewählt, die im Jahr 2000 in Auftrag gegeben wurden. Im Entwurfsmodus können Sie nachträglich Änderungen an Abfragen vornehmen.

10.3.2 Bericht mit Datenquelle verbinden

Damit die in Tabellen oder Sichten zusammengefassten Daten im Bericht dargestellt werden können, ist eine Verknüpfung zwischen Bericht und Datenquelle erforderlich. Die Abfrage aus dem letzten Abschnitt sei unsere Datenquelle für dieses Beispiel. Wenn Sie den Assistenten zur Berichterstellung verwenden, dann wird gleich im ersten Dialogfeld nach der Datenherkunft gefragt.

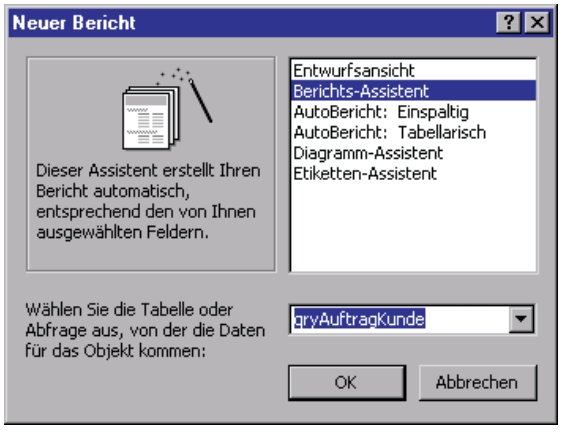


Abbildung 10.6: Berichtsassistent von MS Access

Die Abbildung 10.6 zeigt das Dialogfeld mit der Abfrage *qryAuftragKunde* als gewählte Datenquelle. Es ist jedoch auch denkbar, dass Sie auf den Assistenten verzichten und einen neuen Bericht direkt in der Entwurfsansicht erstellen und die Datenherkunft dort definieren.

1. Lassen Sie sich die Berichteigenschaften anzeigen ANSICHT/EIGENSCHAFTEN.
2. Öffnen Sie die Registerkarte DATEN.
3. Wählen Sie in der Auswahlliste für die Datenherkunft die Abfrage *qryAuftragKunde* (siehe Abbildung 10.7).
4. Schließen Sie das Dialogfeld.

Damit stehen die Felder, welche in der Abfrage aufgelistet wurden, im Bericht zur Verfügung.

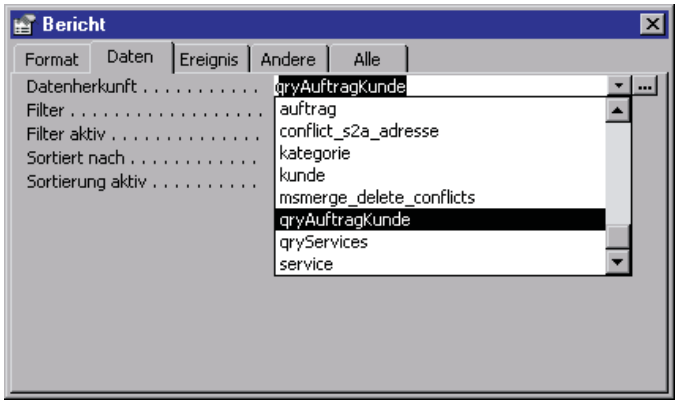


Abbildung 10.7: Auswahl der Datenherkunft für einen Bericht

10.3.3 Felder anordnen

Die Felder aus der Abfrage können ebenfalls per Assistent dem Bericht zugefügt werden. Das entsprechende Vorgehen kann der Abbildung 10.8 entnommen werden. Die effektivste Vorgehensweise ist ein erstes Anordnen der Felder durch den Assistenten und ein späteres Nachbearbeiten im Entwurfsmodus.

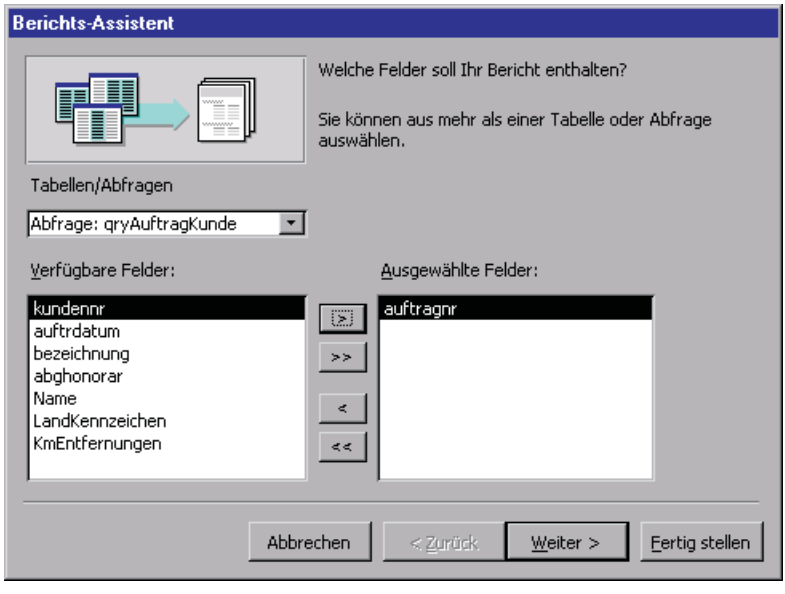


Abbildung 10.8: Auswahl der Felder im Berichtsassistenten

Wollen Sie einen Bericht nachträglich im Entwurfsmodus ändern, um z.B. neue Felder hinzuzufügen, dann können Sie dies mit folgendem Ablauf realisieren:

1. Bericht im Entwurfsmodus öffnen.
2. Klicken Sie in der Tastenreihe (Buttonbar) auf FELDLISTE.
3. Wählen Sie aus der Feldliste das Feld, das Sie einfügen wollen (siehe Abbildung 10.9).

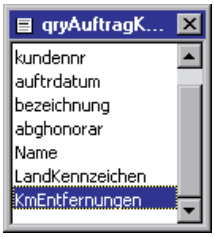


Abbildung 10.9: Die Auswahlliste der Felder

4. Positionieren Sie das Feld im Detailbereich. Im Bild 10.10 wurde dies mit dem Feld *kmEntfernung* getan.

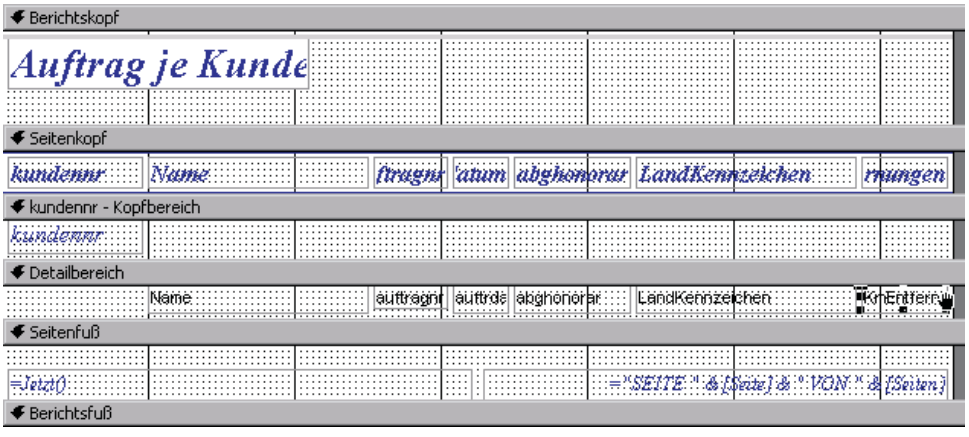
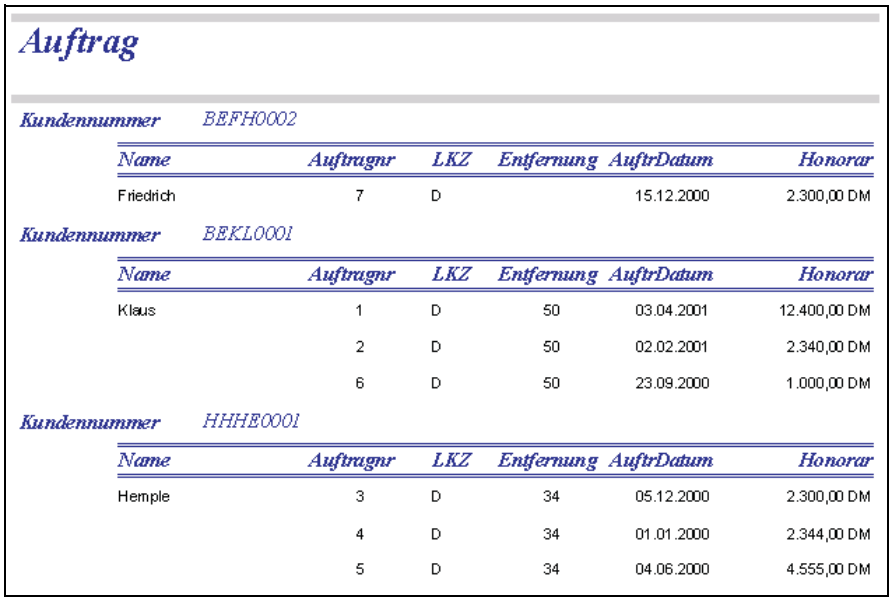


Abbildung 10.10: Rohentwurf des Berichtes

5. Wiederholen Sie den Vorgang für weitere Felder, die eingefügt werden sollen.
6. Beenden Sie den Entwurfsmodus.

Abgesehen von den Abschneidungen bei den Feldbezeichnern ist der Bericht, wie er in Bild 10.10 zu sehen ist, noch recht unübersichtlich. Durch eine geänderte Reihenfolge

der einzelnen Felder im Bericht kann man die Lesbarkeit des Berichtes deutlich erhöhen. Besonders interessante Informationen, wie das abgerechnete Honorar, sollten hervorgehoben werden oder sie sollten zumindest nicht mitten im Bericht erscheinen. Wir wollen die Reihenfolge verändern, so dass sie der Abbildung 10.11 entspricht.



Auftrag

Kundennummer BEFH0002

Name	Auftragnr	LKZ	Entfernung	AuftrDatum	Honorar
Friedrich	7	D		15.12.2000	2.300,00 DM

Kundennummer BEKL0001

Name	Auftragnr	LKZ	Entfernung	AuftrDatum	Honorar
Klaus	1	D	50	03.04.2001	12.400,00 DM
	2	D	50	02.02.2001	2.340,00 DM
	6	D	50	23.09.2000	1.000,00 DM

Kundennummer HHHE0001

Name	Auftragnr	LKZ	Entfernung	AuftrDatum	Honorar
Hemple	3	D	34	05.12.2000	2.300,00 DM
	4	D	34	01.01.2000	2.344,00 DM
	5	D	34	04.06.2000	4.555,00 DM

Abbildung 10.11: Bericht im Ausführungsmodus

Das Honorar wir nun als letztes Feld aufgelistet. Da bereits eine Gruppierung für das Feld *kunde* erstellt wurde, wäre es auch sinnvoll, ein Summenfeld für das Honorar einzufügen. Für jeden Kunden soll die Summe aller abgerechneten Honorare im Bericht erscheinen. Führen Sie dazu die folgenden Schritte aus:

1. Öffnen Sie den Bericht im Entwurfsmodus.
2. Lassen Sie sich das Dialogfeld SORTIEREN UND GRUPPIEREN anzeigen: ANSICHT/SORTIEREN UND GRUPPIEREN.
3. Klicken Sie im Dialogfeld auf das Feld *kundennr*.
4. Geben Sie für Gruppenfuß, bei den Gruppeneigenschaften, JA an (siehe Abbildung 10.12).
5. Im Entwurfsmodus des Berichtes wurde nun ein Fußbereich für die Gruppierung eingefügt.
6. Fügen Sie aus der Toolbox ein Textfeld in den Fußbereich, direkt unter das Feld *abghonorar*.

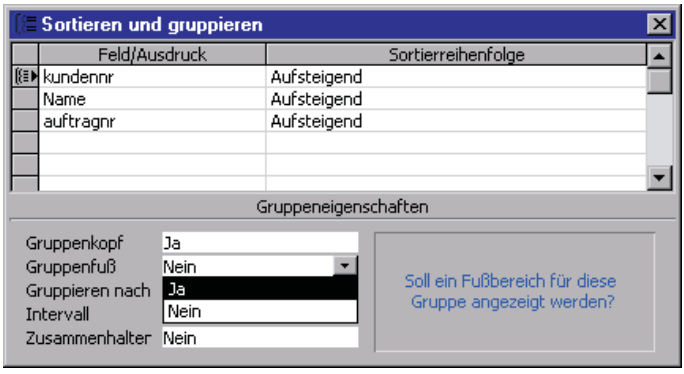


Abbildung 10.12: Gruppeneigenschaften in MS Access

7. Schreiben Sie in das Bezeichnungsfeld den Ausdruck »Summe:«
8. Verbinden Sie das soeben eingefügte Textfeld mit dem Feld *abghonorar*: ANSICHT/EIGENSCHAFTEN/DATEN/STEUERELEMENTINHALT.
9. Geben Sie bei der laufenden Summe »Über Gruppe« an.
10. Zeichnen Sie über das Textfeld einen Summenstrich.

Eine weitere Optimierung wäre ein Ausblenden von Duplikaten. Wenn Sie den Bericht ausführen, werden Sie sehen, dass in jeder Detailzeile der Name des Kunden angegeben wird. Bei mehreren Zeilen je Kunde führt dies zu unerwünschten Mehrfachnennungen des Kundennamens. Über die Feldeigenschaften können wir den Schönheitsfehler beheben. Blenden Sie die Duplikate, wie es auch in Abbildung 10.13 zu sehen ist, aus.

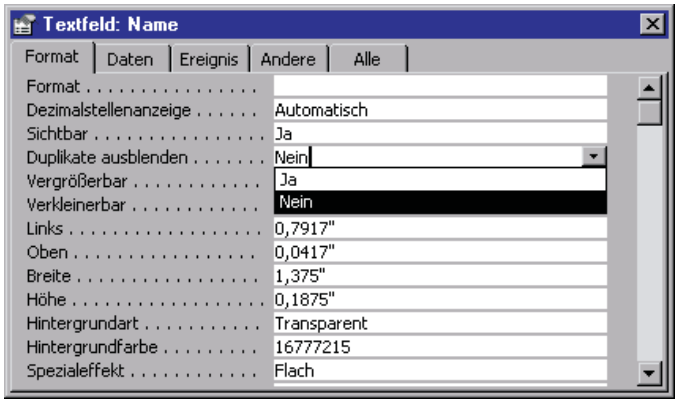


Abbildung 10.13: Ausblenden der Duplikate im Eigenschaftenfenster

10.3.4 Layout bearbeiten

Zum Schluss sollen Formartierungsarbeiten durchgeführt werden. Der Assistent verteilt Felder und Feldbezeichner gleichmäßig auf der verfügbaren Fläche, wobei er nicht auf Abschneidungen und tatsächlichen Größen der Werte achtet. Deshalb müssen die Größen der im Bericht erscheinenden Steuerelemente manuell angepasst werden. Nachträglich eingefügte Felder sind vielleicht nicht nach Ihren Wünschen ausgerichtet. Die Berichtsüberschrift sollte eventuell verändert werden. All dies sind Aufgaben, die das Layout betreffen und nachträglich im Entwurfsmodus durchgeführt werden. Wählen Sie dazu das zu verändernde Steuerelement aus und verändern Sie die Eigenschaften oder manipulieren Sie es direkt mit dem Cursor.

Auftrag je Kunde

KundenNr	Name	LKZ	Entfernungen [km]	AuftrNr	AuftrDatum	abgerechnetes Honorar
BEFH0002	Friedrich	D		7	15.12.2000	2.300,00 €
					Summe:	**2.300,00 €**
BEKL0001	Klaus	D	50	1	03.04.2001	12.400,00 €
				2	02.02.2001	2.340,00 €
				6	23.09.2000	1.000,00 €
					Summe:	**15.740,00 €**
HHHE0001	Hemple	D	34	3	05.12.2000	2.300,00 €
				4	01.01.2000	2.344,00 €
				5	04.06.2000	4.555,00 €
					Summe:	**9.199,00 €**

Abbildung 10.14: Auftragsbericht in MS Access

Der Bericht »Auftrag je Kunde« könnte schließlich ein Aussehen haben, wie es in der Abbildung 10.14 dargestellt ist. Die Duplikate sind sowohl für den Namen, als auch für die *KundenNr*, *LKZ* und die *Entfernung* ausgeblendet. Einheiten sind je nach Bedarf im Tabellenkopf oder zusammen mit den Werten darzustellen. Felder gleicher Art stehen nebeneinander. Die Summenwerte sind hervorgehoben, damit sie schnell auf dem Bericht auszumachen sind. Nicht abgebildet sind Seitenzahl, Datum und Gesamtsumme für das abgerechnete Honorar, die im Kopf- und Fußbereich anzuordnen sind. Beachten Sie außerdem die Richtlinien aus Abschnitt 10.1.

10.3.5 Überprüfen der Werte

Bevor Sie den Bericht ausliefern oder ihn produktiv einsetzen, müssen Sie die ausgewiesenen Werte überprüfen. Verwenden Sie für den Test Datenwerte, die den realen Werten entsprechen oder zumindest sehr nahe kommen. Führen Sie den Test auch mit größeren Datenmengen aus, damit Seitenwechsel und mögliche Überläufe simuliert werden. Achten Sie insbesondere auf Summen, umgerechnete und zusammengefasste Werte.

Gibt es Unstimmigkeiten, so verfolgen Sie den Fehler schrittweise von den Daten in der Datenbank, über die SQL-Anweisung, welche die Daten vorbereitet, bis hin zur Logik im Bericht. Eine mögliche Fehlerquelle sind unterdrückte Zeilen im Detailbereich. Es sei nochmals, auch auf dem Hintergrund einer zusätzlichen Fehlerquelle, darauf hingewiesen, dass eine Verdichtung der Datensätze nicht erst durch Funktionen im Berichtsgenerator erfolgen sollte. Nehmen Sie eine Verdichtung bereits mit *Aggregatfunktionen* bei den verwendeten SQL-Anweisungen vor.

10.4 3D-Säulendiagramm

Mit Diagrammen bringen Sie ein wenig Salz in die lasche »Zahlensuppe«. Besonders für Präsentationen, bei denen die Zuhörer nur wenig Zeit haben, um aus den Zahlenwerten Trends abzulesen, kann ein Diagramm eine große Hilfe sein. Berichtswerkzeuge stellen dem Entwickler in der Regel mehrere Diagrammtypen zur Verfügung. In diesem Abschnitt wird demonstriert, wie Sie mit MS Access ein 3D-Säulendiagramm erstellen können. Diagramme gehören auch zur Kategorie der Berichte und benötigen wie diese eine Datenquelle. Für das folgende Beispiel verwenden wir die Abfrage *qryAuftragKunde*, welche bereits im letzten Abschnitt erstellt wurde.

1. Starten Sie den Diagramm-Assistenten, EINFÜGEN/BERICHT/DIAGRAMM-ASSISTENT.
2. Wählen Sie die Abfrage *qryAuftragKunde* als Objekt der Datenherkunft.
3. Fügen Sie die Felder *name* und *abghonorar* dem Diagramm zu.
4. Wählen Sie 3D-Säulendiagramm als Diagrammtyp.
5. Bestimmen Sie das Layout für das Diagramm.
6. Vergeben Sie einen Namen für das Diagramm.
7. Geben Sie an, ob das Diagramm eine Legende enthalten soll oder nicht.
8. Klicken Sie auf FERTIG STELLEN.

Nachdem der Assistent den Rohentwurf des Diagramms erstellt hat, können Sie nun im Entwurfsmodus das Diagramm optimieren. Klicken Sie dazu auf das Diagrammobjekt mit der rechten Maustaste und wählen Sie im Kontextmenü DIAGRAMMOPTIONEN. Das geöffnete Dialogfeld ist in Abbildung 10.15 zu sehen. Hier können Sie die Beschriftung der Achsen ändern, die Legende ausblenden, die Datenbeschriftung neu definieren oder die Datentabelle anzeigen lassen. Geben Sie eine Zeichenfolge für die Titelbezeichnung sowie die Achsenbeschriftung an.

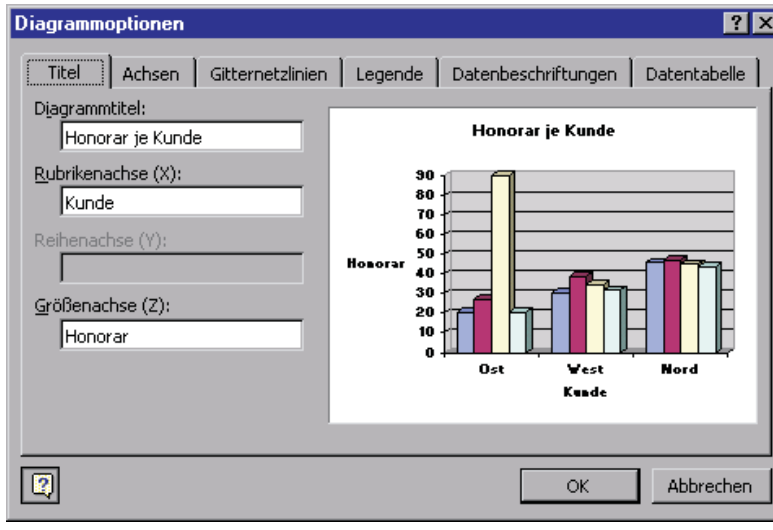


Abbildung 10.15: Einstellung der Diagrammoptionen

Nach weiteren Anpassungen bei der Achsenbeschriftung und der Gesamtgröße hat das Diagramm ein Aussehen, wie es in Abbildung 10.16 dargestellt ist.

Es können globale Diagramme erstellt werden, bei denen sämtliche Datensätze aus der Datenherkunft berücksichtigt werden. Für jeden Datensatz kann jedoch auch ein separates Diagramm vorgegeben werden. Beim Navigieren durch die Datensätze wird dann immer ein anderes Diagramm aufgebaut. Zusätzlich können Sie Diagramme aus anderen Programmen als Objekt über die OLE-Schnittstelle einbinden.

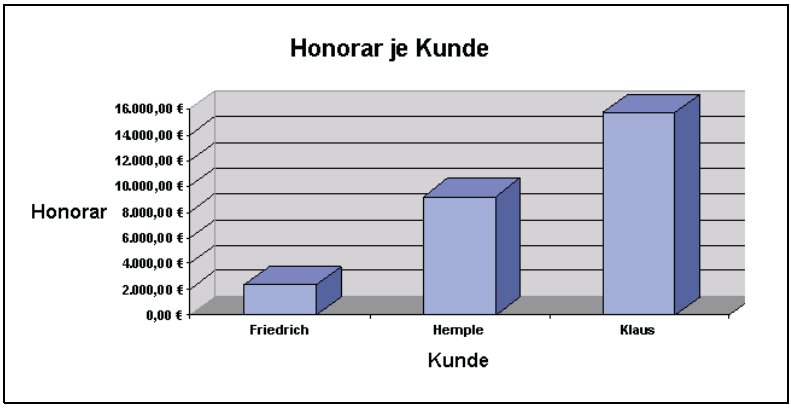


Abbildung 10.16: D-Säulendiagramm für das Honorar

10.5 Berichte verteilen

Wenn Sie Berichte drucken und an Benutzer innerhalb und außerhalb Ihres Unternehmens verteilen, kann die Verwendung von Bericht-Snapshots Zeit und Geld sparen. Sie müssen die gedruckten Microsoft Access-Berichte nicht fotokopieren und versenden, sondern können sie elektronisch verteilen und veröffentlichen, entweder über E-Mail oder mittels eines Webbrowsers, z.B. Microsoft Internet Explorer. Benutzer erhalten leicht und schnell Onlinezugriff auf die Berichte und können die für sie erforderlichen Berichtsseiten ausdrucken. Dies ist vor allem dann von Vorteil, wenn Ihre Berichte Farb- und Bildelemente, z.B. Diagramme und Grafiken enthalten (siehe Abbildung 10.17).

Sie können einen Bericht-Snapshot mit MS Access erstellen. Um einen Bericht-Snapshot anzuzeigen, zu drucken, zu speichern, zu veröffentlichen, zu verteilen oder zu archivieren, ist jedoch keine Access-Lizenz erforderlich. Stattdessen kann eine Kombination aus Snapshot Viewer und anderen Programmen, wie Windows Explorer, einem E-Mail-Programm oder einem Webbrowser, z.B. Internet Explorer, verwendet werden.

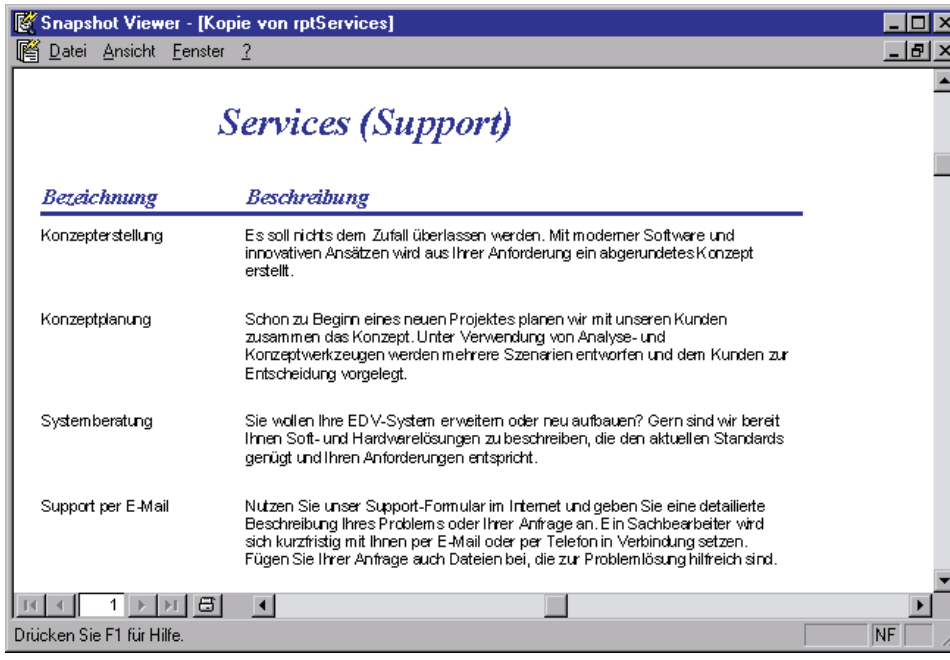


Abbildung 10.17: Bericht im Snapshot Viewer

11 E-Commerce und Web

Die neuen Funktionalitäten, die dem Unternehmen zur Datenanalyse und zur Web-Publikation mit modernen Datenbanksystemen an die Hand gegeben werden, sind Schwerpunkt in diesem Kapitel. Neue Technologien wie Data Warehousing oder OLAP-Datenbanken werden vorgestellt. Firmen werden immer stärker zu globalen Unternehmen und die gesammelten Daten sollen auch global abrufbar sein. Die Voraussetzungen und die technischen Schritte zur Umsetzung einer Datenbankanbindung im Internet werden Sie in diesem Kapitel kennen lernen. Auch hier ist die Beispieldatenbank wieder Mittel zum Zweck der illustrativen Darstellung.

11.1 Die Sammlung der Unternehmerdaten: Data Warehousing

Datenbanken sind eine wesentliche Grundlage für die betriebliche Informationsversorgung. Aber meistens existiert in einem Unternehmen nicht nur eine Datenbank, sondern mehrere. So können z.B. Kundendaten in verschiedenen Datenbanken enthalten sein, aber in einer gemeinsamen Auswertung benötigt werden. Besonders bei dezentraler Datenhaltung kann es erforderlich sein, auf verschiedene Datenquellen zurückzugreifen und sie gemeinsam auszuwerten.

11.1.1 Bekannte Probleme in den Unternehmen

Ein wesentliches Problem in den Unternehmen sind *heterogene EDV-Strukturen*. In den verschiedenen Bereichen eines Unternehmens existieren häufig unterschiedliche Datenbanken. Je nach Vorlieben der Entscheider, sind die Systeme über Jahre gewachsen oder auch in Teilbereichen komplett erneuert worden. Der Hauptaugenmerk wurde auf die Abdeckung spezifischer Anforderungen der jeweiligen Bereiches gelegt. Übergreifende Auswertungen sind nur bedingt und zeitaufwändig durchführbar. Teilweise behindern darüber hinaus unterschiedliche firmeninterne Richtlinien und Ordnungsbegriffe die Zusammenführung der Daten.

Den Anwender in den Fachabteilungen fehlen häufig Mittel und Kenntnisse, um aus ihren Datenbeständen die entsprechenden Kenntnisse zu ziehen. Komfortable grafi-

sche Benutzeroberflächen zur Formulierung der Abfragen stehen den Fachabteilungen nicht zur Verfügung, so dass die gewünschten Analysen nur mit Programmierkenntnissen durchführbar sind. Nicht immer haben die einzelnen Abteilungen das Glück, entsprechend qualifiziertes Personal zum Programmieren solcher individuellen Problemstellungen zur Verfügung zu haben. Deshalb sind sie auf die EDV-Abteilung oder externe Anbieter angewiesen.

Die vorhandene EDV-Infrastruktur wird bei Erstellung der Auswertungen sehr stark belastet. Die Systeme sind meist auf effiziente Bearbeitung einzelner Transaktionen optimiert, die das Tagesgeschäft beschleunigen sollen (z.B. Rechnungs- und Auftragserstellung). Bei diesen Transaktionen wird nur auf geringe Datenmengen in wenigen Tabellen zugegriffen. Analysen und Berichte erfordern aber den Zugriff auf eine große Menge von Daten und oft auf eine Vielzahl von Tabellen des relationalen Systems.

Bei den Analysen spielen immer häufiger verdichtete (aggregierte) Daten und Vergleiche über längere Zeiträume eine Rolle, um Trends erkennen zu können. Ein Dienstleistungsunternehmen, will beispielsweise wissen, wie ein spezieller Service von den Kunden angenommen wurde. Dabei sind die Daten vom erstmaligen Anbieten des Service bis zur aktuellen Analyse relevant. Historische Daten werden jedoch oft von operativen Systemen auf kostengünstigere, langsame Datenträger ausgelagert, auf die nicht online zugegriffen werden kann. Damit fehlen die erforderlichen historischen Daten.

Tatsache ist, dass in operativen Systemen permanent neue Geschäftsaktivitäten verarbeitet werden. Frühere Auswertungen können zu einem späteren Zeitpunkt oft nicht mehr nachvollzogen werden, da die erforderlichen historischen Daten nicht verfügbar sind. In operativen Systemen müssen die verschiedenen Geschäftsaktivitäten sofort zu einer Aktualisierung der Daten führen. Eine Analyse, die auf diese Daten zurückgreift, kann daher unter Umständen schon nach kurzer Zeit nicht mehr nachvollzogen werden, da die zugrundeliegenden Werte in den operativen Systemen durch neue Transaktionen geändert wurden.

Viele Unternehmen haben also riesige, historisch angewachsene, Datenbestände, die sie jedoch nicht oder nur begrenzt nutzen können, um Informationen über Profitabilität einzelner Produkte oder Dienstleistungen zu erhalten.

11.1.2 Sinn und Zweck von Data Warehouse

Die Beseitigung der beschriebenen Probleme kann mit einer Data Warehouse-Lösung nicht vollständig garantiert werden, aber sie trägt dazu bei. Durch ein *Data Warehouse* sollte es möglich sein, den verschieden Abteilung und auch den einzelnen Mitarbeitern Schlüsselinformationen zugänglich zu machen, die sie zur Optimierung von internen Geschäftsabläufen benötigen. Ein Data Warehouse muss so aufgebaut sein, dass es aktuelle und nachvollziehbare Informationen in einer effektiven Art und Weise zur Verfügung stellt.

Die Verfügbarkeit problemadäquater Informationen für ein gezieltes Marketing oder die frühzeitige Erkennung von Chancen und Risiken wird für Unternehmen zunehmend an Bedeutung wachsen. Neben der Kontrolle von Entscheidungen durch Standardauswertungen, werden Informationen für eine gezielte Ausgestaltung der Marketingstrategie immer wichtiger. Neue Zusammenhänge in den Datenbeständen müssen frühzeitig erkannt werden. Welche Kunden, tragen bei welchen Bedingungen, zu besonders hohem Umsatz bei? Welche Zulieferer beeinflussen den Produktionsablauf, durch verspätete Lieferzeiten und schlechte Qualität negativ? Welche Mitarbeiter arbeiten besonders effizient? Diese Fragen sollen zeigen, dass nahezu alle Abteilungen ein Interesse an Schlüsselinformationen haben, die aus den *Datenanalysen* gewonnen werden können.

Kern ist eine Datenbank mit allen entscheidungsrelevanten Informationen. Diese Datenbank ist eine Zusammenführung der Datenbanken, die für die Abwicklung der operativen Geschäftsprozesse eingesetzt werden, ergänzt um Daten aus externen Quellen. Verwendet werden zunächst Daten, die für die Analyse erforderlich sind.

Ergänzt wird diese Datenbank um Softwarewerkzeuge, mit denen die Daten in die Data Warehouse-Datenbank eingestellt werden und mit denen diese Daten danach abgefragt und analysiert werden können.

11.1.3 Anforderungen

Im Vergleich zu den operativen Datenbeständen der Administrations- und Dispositionssysteme, muss ein Data Warehouse grundsätzlich andere Anforderungen erfüllen. Diese sind zu verschiedenen Regeln für ein Data Warehouse zusammengefasst worden:

1. Die Entscheidungen in einem Unternehmen basieren im Wesentlichen auf Informationen über z.B. Kunden oder Produkte und weniger auf Informationen über die innerbetrieblichen Prozesse. Die *Speicherung der Informationen* in einem Data Warehouse sollte sich daher an den Subjekten eines Unternehmens, z.B. den Kunden, orientieren und nicht an den innerbetrieblichen Abläufen. Daten, die nicht der Entscheidungsunterstützung dienen, werden nicht in das Data Warehouse aufgenommen.

2. Das Data Warehouse ist eine Zusammenfassung verschiedenster Datenbanken. Durch *Struktur- und Formatvereinheitlichung* müssen zusammengehörige Daten bereinigt und in einer übergreifenden Datenstruktur abgebildet werden.

3. Mit der Unveränderbarkeit der Daten, soll die *Reproduzierbarkeit* der Analyseergebnisse gewährleistet werden.

4. Data Warehouse-Daten können vom Anwender nicht gelesen werden. In ein Data Warehouse werden neue Daten hinzugeladen oder andere archiviert. *Nie werden vorhandene Daten verändert* – auch nicht von anderen Programmen.

5. Alle Daten erhalten den *Bezug zu einem Zeitraum*, für den sie gültig sind. Damit können historische Daten, parallel zu aktuellen Informationen gespeichert und verwendet werden.

6. Eine wesentliche Anforderung sind *flexiblere Analysemöglichkeiten*. Dies ist im Interesse der Fachabteilung und der internen EDV. Durch einfach zu bedienende grafische Oberflächen können die Analysen in der Fachabteilung selbst erstellt werden. Papierberge und entsprechende Kosten können abgebaut werden, indem Auswertungen bei Bedarf online erzeugt werden.

7. Durch eine umfassende Informationsbasis, in die auch externe Daten einbezogen werden, sollen fundiertere Entscheidungen ermöglicht werden. Häufig finden sich in unterschiedlichen Berichten nicht vergleichbare Informationen, da diese Berichte auf unterschiedlichen Datenbanken in den verschiedenen Bereichen eines Unternehmens beruhen. Dies kann vermieden werden, wenn alle Bereiche eines Unternehmens auf eine *gemeinsame, bereinigte Datenbasis* zugreifen.

8. Für Data Warehouse-Umgebung sind im Allgemeinen große und komplexe Systeme erforderlich. Leistungstarke Server und Netzwerke mit ausreichend hoher Bandbreite sind die Grundlage für eine *solide Hardware-Architektur*. Außerdem sind für die Verwaltung eines solch komplexen Systems Werkzeuge für die Verwaltung erforderlich. Mit geeigneten Verwaltungswerkzeugen können frühzeitig Probleme, wie übermäßig hoher Ressourcenverbrauch oder Überschreiten von Schwellenwerten bei der Prozessorauslastung, erkannt werden.

11.1.4 Architektur

Bei einem Data Warehouse können zwei wesentliche Bereiche unterschieden werden. Der Bereich der Datenbereitstellung und der der Informationsgewinnung.

Als Grundlage müssen die verschiedensten operativen Datenbestände und weitere externe Daten (z.B. Marktforschungsdaten) zu einem gemeinsamen, konsistenten Datenbestand zusammengeführt werden. Ausgangsbasis können relationale Datenbanken oder sonstige Dateien sein.

Aufgrund der Heterogenität der Ausgangsdaten, verursacht der Bereich der Datenbereitstellung meistens den wesentlichen Aufwand bei einer Data Warehouse-Entwicklung. Die Erledigung dieser Aufgabe erfüllt, wie dies auch in Abbildung 11.1 gezeigt ist, der *Einfüge-Manager*. Diese Komponente kann eine Kombination aus verfügbaren Standardlösungen und individuell zugeschnittenen Softwarelösungen (gespeicherte

Prozeduren, Skripte und weitere Programme) sein. Die Aufgaben des Einfüge-Managers sind im Einzelnen:

- Schnelles Einfügen in die Datenbank,
- Konvertierung der Daten,
- Ablaufkontrolle,
- Prüfung und Fehlerauswertung.

Die umgewandelten Daten werden in einer Datenbank, dem Data Warehouse im engeren Sinne, abgelegt. In den meisten Fällen handelt es sich hier um eine von den operativen Systemen getrennte Datenbank.

Auf diese kann dann für die gewünschten Auswertungen und Analysen über einen *Abfrage-Manager* zugegriffen werden. Der Abfrage-Manager hat die Aufgabe, die Ausführung der Anwenderabfragen zu planen und diese zu den entsprechenden Tabellen zu lenken. Als Entscheidungswerkzeuge kommen viele Möglichkeiten in Betracht. Konventionelle Reportgeneratoren könnten zum Einsatz kommen, mit denen vordefinierte Auswertungen automatisch zu vorher festgelegten Zeitpunkten oder ad hoc erzeugt werden. Eine andere Möglichkeit besteht in der weiteren Auswertung und Verarbeitung der Data Warehouse-Daten mit Spreadsheet-Programmen, wie z.B. Excel. Damit können zusätzliche statistische Berechnungen auf den Data Warehouse-Daten ausgeführt werden. OLAP-Werkzeuge und Datenanalysen werden im Abschnitt 11.2 näher beschrieben.

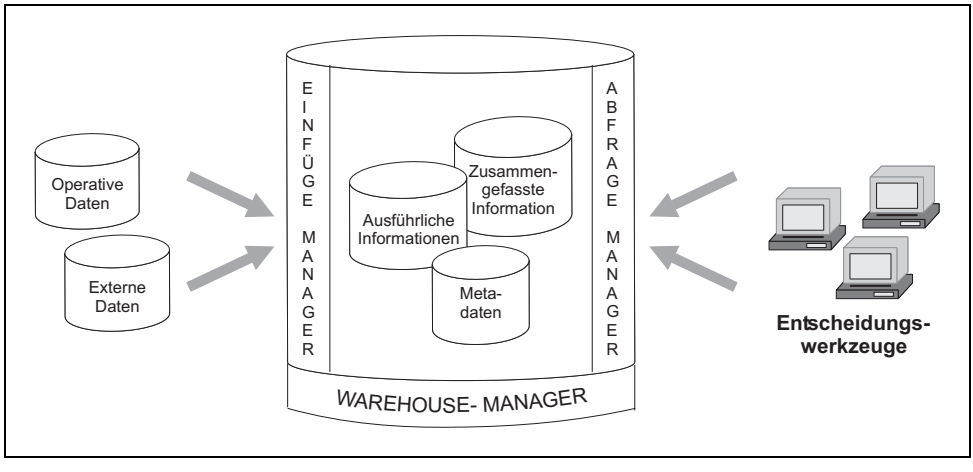


Abbildung 11.1: Die Architektur beim Data Warehousing

Data Mining ist die neueste Auswertungstechnologie. Bei den bisher genannten Auswertungsverfahren sind zumindest die Dimensionen der Analyse vorgedacht und die Zusammenhänge zwischen diesen Strukturen sind bekannt oder werden unterstellt. Die Analyse der Informationen erfolgt intuitiv und weitgehend manuell durch den Benutzer. Bei *Data Mining* sollen automatisch neue Muster und Zusammenhänge in den Daten erkannt und der Benutzer darauf aufmerksam gemacht werden.

Der Warehouse-Manager ist die Systemkomponente, die alle notwendigen Operationen zur Unterstützung des Verwaltungsprozesses für das Data Warehouse durchführt. Ausführliche Informationen werden durch Aggregationen zusammengefasst, damit die Abfrageleistung beschleunigt wird. Damit beim Hinzufügen von neuen Daten nicht die zugehörige Zusammenfassung der Daten überarbeitet werden muss, bedient sich der Warehouse-Manager der *Metadaten*, durch welche die Erstellung der Zusammenfassung der Daten gesteuert wird.

11.1.5 Datenbereitstellung

Wie bereits geschildert, werden im ersten Schritt die Daten der bestehenden Datenbanken und weitere externe Daten einmalig oder periodisch übertragen. Da viele Daten zeitlich relativ stabil sind, reicht es meist aus, die Daten, welche seit dem letzten Abzug verändert wurden oder neu hinzugekommen sind, einzufügen.

Besondere Bedeutung kommt der Transformation der Ausgangsdaten zu, da hier die Qualität des Datenbestandes und damit der späteren Analysen entscheidend festgelegt wird. Aus den unterschiedlichen Daten muss ein einheitlicher Datenbestand erzeugt werden. Die Felder der operativen Datenbanken müssen in die Felder der Data Warehouse-Datenbank transformiert werden. In den verwendeten Werkzeugen werden hierfür Regeln definiert, nach denen die Data Warehouse-Daten gewonnen werden. Der Transformation liegt ein Schema zugrunde, welches die Form bestimmt, in das die Daten umgeformt werden. Drei gebräuchliche Varianten sind:

- Star-Schema,
- Snowflake-Schema,
- Starflake-Schema.

Es handelt sich um Datenbankschemata, welche die Daten so strukturieren, dass die Abfragen zur Entscheidungshilfe unterstützt werden. Ein Beispiel für das Star-Schema sehen wir in Abbildung 11.2:

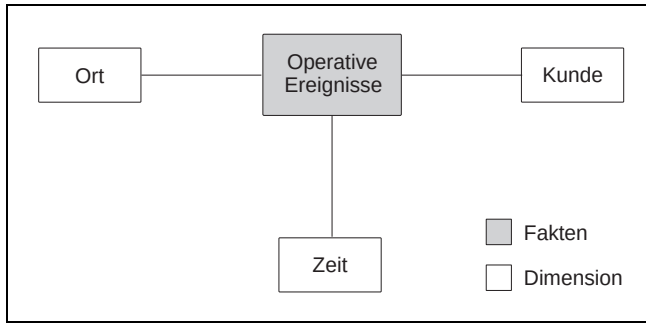


Abbildung 11.2: Das Star-Schema

Die Basis stellen so genannte *Faktentabellen* dar. Hierin enthalten sind die Daten für operative Ereignisse, wie Verkaufsaktionen oder Kundenaktivitäten. Die Fakten sind der wesentliche Bestandteil des Datenaufkommens eines typischen Data Warehouses. Die meisten Abfragen durchsuchen zuerst die Daten in der Faktentabelle, damit die Anzahl der zu sichtenden Zeilen verringert wird. Durch die Verringerung der gesamten zu sichtenden Datenmenge kann die zur Ausführung einer Abfrage benötigte Zeit verringert werden.

Die Basisinformationen in den Faktentabellen werden ergänzt durch Referenzinformationen, die verwendete werden, um die Basisereignisse zu analysieren. Die Referenztabellen finden wir in den *Dimensionstabellen*. Im Star-Schema aus Abbildung 11.2 bilden Ort, Zeit und Kunde die Dimensionen. Es sollte damit gerechnet werden, dass sich die Dimensionsdaten regelmäßig ändern. Das Data Warehouse sollte auf einen geringen Änderungsaufwand für diese Daten zugeschnitten sein.

Ein wichtiger Aspekt beim Entwurf des Datenbankschemas ist das Erkennen der Fakten- und der Dimensionsdaten. Dafür ist es zwingend notwendig, sich mit dem Unternehmensmodell und der Struktur der operativen Daten vertraut zu machen. Grundlegende Geschäftsabläufe wie Kundenereignisse oder Kontotransaktionen sind potentielle Entitäten, die für die Faktentabellen in Frage kämen. Je nach Fragestellung bzw. Zielvorgaben für die Analyse, ist es möglich, dass die gleiche Entität einmal als Dimension und ein anderes Mal als Fakt definiert wird. Der Entwurfsprozess bedarf neben der Kenntnis einiger Richtlinien auch eines gewissen Maßes an Erfahrung im Umgang mit Data Warehouses.

Es werden Abfragen verwendet, bei denen eine räumliche Darstellung der Informationen sinnvoll ist. Jede Dimension beansprucht dabei eine Achse und die Werte im Raum entsprechen den Zustandstransaktionen.

Eine mehrdimensionale Darstellung der Produkte nach Zeit und pro Kunde finden wir in der Abbildung 11.3. Wir erhalten einen Würfel (Cube), der eine Menge von Daten beinhaltet. Interessiert uns beispielsweise die Menge der verkauften Produkte an

einem Tag je Kunde, so gehen wir von der dreidimensionalen Ansicht in die zweidimensionale. Als Ergebnis erhalten wir die Summe aller bestellten Produkte je Kunde bei einer konstanten Zeitangabe.

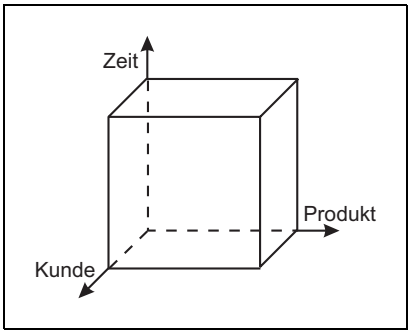


Abbildung 11.3: Mehrdimensionale Datendarstellung

11.1.6 Metadaten

Wesentliche Grundlage für diese Systeme sind *Metadaten*. Metadaten sind Daten über Daten. Metadaten kommen nicht nur bei Data Warehouse vor, sie sind in der Datenverarbeitung häufig vorzufinden.

Die Metadaten beschreiben die Struktur und die Zusammenhänge eines Systems. Die Metadaten bilden damit die Grundlage dafür, komplexe Systeme verstehen, warten und weiterentwickeln zu können. Wie bereits angedeutet, wäre es ohne Metadaten sehr aufwändig, zusammengefasste Datenmengen aufgrund von Änderungen an der operativen Datenbanken zu aktualisieren. Es muss nachvollziehbar sein, welche Data Warehouse-Datenfelder davon betroffen sind. Auch Transformationsregeln sind als Metadaten abgelegt und stehen den Zugriffswerkzeugen zur Verfügung. Im Data Warehouse können Metadaten für folgende Bereiche eingesetzt werden:

- Verwalten von Daten,
- Datenumwandlung und Einfügen,
- Erzeugen von Abfragen.

Die Metadaten sind wesentliches Hilfsmittel für den Benutzer, um das System verstehen und zielgerichtet nutzen zu können. Beispiele hierfür sind Informationen über existierende Standardberichte/Berichtsvorlagen, fachliche Berechnungsvorschriften und Begriffslexika.

Die Metadaten können in einer separaten Datenbank abgelegt sein oder in der gleichen Datenbank abgelegt werden, wie die entscheidungsrelevanten Daten.

11.1.7 Aggregationen

Ein wesentliches Merkmal des Data Warehouse ist die *Aggregation*. Durch Datenzusammenfassung kann eine kosteneffektive Abfrageleistung ermöglicht werden. Aggregationen erlauben es ferner, Trends innerhalb der Datenbestände auszumachen, die bei Betrachtung der Einzelheiten unter Umständen schwer erkennbar wären.

Die Idee hinter den Aggregationen ist für die Analyse notwendige Teilmengen der Daten nach gewissen Kriterien zusammenzufassen. Dies soll anhand der Beispieldatenbank erklärt werden:

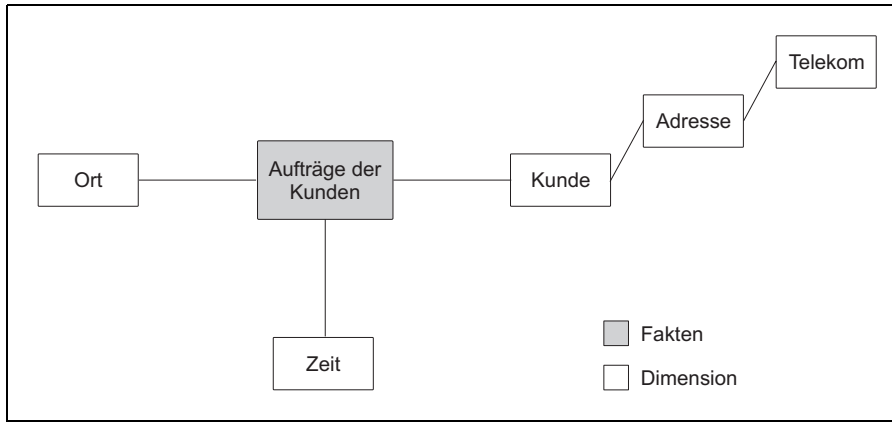


Abbildung 11.4: Beispielschema mit Aggregationsbedarf

Beispiel 11.1:

Die Abbildung 11.4 zeigt eine Teilmenge der Beispieldatenbank im Schema eines Data Warehouse. Die Unternehmensführung steht vor der Entscheidung, das Serviceangebot zu erweitern. Der Kunde soll zukünftig per SMS auf wichtige unternehmensrelevante Termine hingewiesen werden. Zur Entscheidungsfindung sind nun einige Fragen vorab zu klären: Wie viele Kunden sind Teilnehmer am Mobilfunknetz? Gehören die Kunden, die mobil zu erreichen sind, zu der umsatzstarken Zielgruppe? In welchen Mobilfunknetzen sind diese Kunden zu Hause? Die für die Analyse benötigten Informationen stecken in mehreren Dimensionen. Es wären mehrere Schritte notwendig, um die »umsatzstarken« Kunden mit Zugang zum Mobilfunknetz herauszufiltern. Diese Schrittfolge wäre auch später bei Analysen zur Effektivität dieses Dienstes notwendig. Eine vorbereitete Zusammenfassungstabelle, die alle Handybesitzer enthält, kann die Struktur etwas vereinfachen und damit Rechenzeit einsparen.

Durch sinnvolle Aggregationen sind die schnellen Abfragezeiten bei der Analyse möglich.

11.2 Analytische Abfrage der Daten mit OLAP

Die Struktur und die Daten, die ein Data Warehouse liefert, eignen sich hervorragend zu einer weiteren Technik in der Informationsgewinnung dem *Online Analytical Processing* (OLAP). Dieser Begriff wurde 1993 von *E.F.Codd* in seiner Abhandlung »Providing OLAP to User-Analysts: An IT Mandate« geprägt.

OLAP ist nicht zwangsläufig an ein Data Warehouse gebunden, aber OLAP und ein Data Warehouse ergänzen sich sehr gut. Das Data Warehouse stellt eine bereinigte, konsistente Datenbasis zur Verfügung. Auf dieser können OLAP-Werkzeuge aufsetzen, um dem Benutzer die entsprechenden Auswertungsmöglichkeiten zur Verfügung zu stellen.

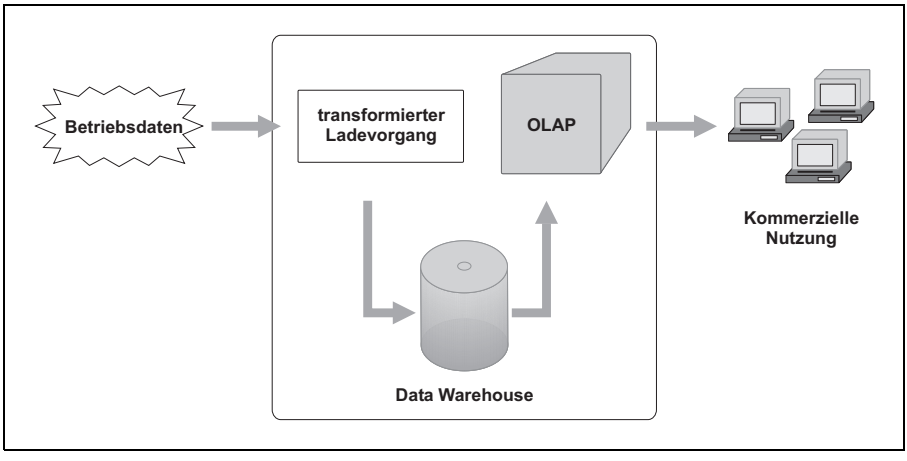


Abbildung 11.5: Datenverarbeitung mit OLAP und Data Warehouse

OLAP-Systeme bestehen aus Daten, Diensten und weiteren Komponenten, wie OLAP-Server oder Präsentationswerkzeuge. Die Dienste und Komponenten arbeiten mit multidimensionalen Abfragen und ermöglichen Analysen großer Datenmengen in Echtzeit. Häufig gestellte Fragen, wie z.B. »Bei welcher Niederlassung haben die meisten Kunden, im ersten Quartal des laufenden Jahres, einen bestimmten Service beauftragt?«. Die Abfragen werden bereits vor der Analyse erstellt, um die Bearbeitungszeit bei der Analyse zu minimieren.

11.2.1 Voraussetzungen für das Arbeiten mit OLAP

Es gibt einige Voraussetzungen für das OLAP-System, deren Erfüllung für sinnvolles Arbeiten notwendig sind.

- Die geforderte Antwort des Systems muss schnell erfolgen.
- Die Analyse vorhandener Datenbestände soll anwenderfreundlich gestaltet sein.
- Die Datenbestände sollen im gemeinsamen Zugriff für mehrere Benutzer verfügbar sein.
- OLAP-Datenbanken sind multidimensional.
- OLAP-Datenbanken besitzen die Fähigkeit, aus Daten Informationen zu erzeugen.

OLAP ermöglicht die mehrdimensionale Analyse betriebswirtschaftlicher Variablen, z.B. Umsatz, Gewinn, nach verschiedenen Kriterien, z.B. Kunden oder Regionen. Die typische Darstellung dieser mehrdimensionalen Sicht ist der Olap-Würfel, der auch gleichzeitig als Synonym für OLAP-Datenbanken benutzt wird. Die Kanten des Würfels sind die *Analysekriterien* oder *Dimensionen*. Die Zellen des Würfels enthalten die *Analysevariablen*.

Für die Dimensionen werden verschiedene Aggregationsstufen definiert, also z.B. Monat, Quartal, Jahr für die Zeit oder Bundesland, Land, Kontinent für die Region. Entsprechend liegen auch die Kennzahlen in den verschiedenen Verdichtungsstufen vor.

Das Wesentliche an dem Modell ist, dass die Daten in mehrdimensionalen Strukturen abgelegt sind. Mit dem OLAP-Würfel lässt sich dies anschaulich darstellen. Es ist durchaus möglich, dass mehr als drei Dimensionen zur Beschreibung der Daten vorhanden sind.

Für konkrete Auswertungen sind schon drei Dimensionen zu viel, weil nicht alle Werte sichtbar sind. Bei mehr als zwei Dimensionen lassen sie sich in einer Auswertung nicht sinnvoll darstellen. In der Visualisierung bleibt auch OLAP wie normale Spreadsheets auf zwei Dimensionen beschränkt. Gegenüber den bisherigen Auswertungen und Spreadsheet-Programmen, werden die Daten, trotz der beschränkten Darstellungsmöglichkeiten, nach mehreren Dimensionen beschrieben, was wesentlich flexiblere Auswertungsmöglichkeiten zur Folge hat.

11.2.2 Beispieldatenbank als OLAP-Datenbank

Ein besonderes Plus bei OLAP ist die Fähigkeit, Daten aus verschiedenen Quellen zu bündeln, um umfassende Aussagen treffen zu können. Die Daten aus unserer Beispieldatenbank werden deshalb mit weiteren Daten eines Dienstleistungsunternehmens ergänzt. Es handelt sich um Kundendaten, die im Excel-Format vorliegen, und die für ein Kundenprofil benötigt werden. Die folgende praktische Anleitung zeigt in mehreren Schritten, wie die Daten aufbereitet werden und wie Sie schließlich zu Ihren kundenspezifischen Aussagen kommen.

Zusammenführen der Daten

Die für die Analyse benötigten Daten sollen zunächst in einer Access-Datenbank zusammengeführt werden. Ebenso könnte für diesen Zweck eine beliebige andere Datenbank verwendet werden.

1. Starten Sie Access und erstellen Sie eine neue Datenbank mit dem Namen *kundenprofil*.
2. Importieren Sie in diese Datenbank die Daten aus der Excel-Arbeitsmappe (siehe Begleit-CD im Verzeichnis: \BEISPIELDATEN\OLAP\KUNDENPROFIL.XLS) und aus der Beispieldatenbank die Tabellen *kunde* und *adresse*.
3. Sichern Sie die Daten und schließen Sie MS Access.

Datenquellen einrichten

1. Erstellen Sie ODBC-Verbindungen für die Access-Datenbank.
2. Starten Sie den Analysis-Manager über START\PROGRAMME\MICROSOFT SQL SERVER\ANALYSIS SERVICES\ANALYSIS-MANAGER.

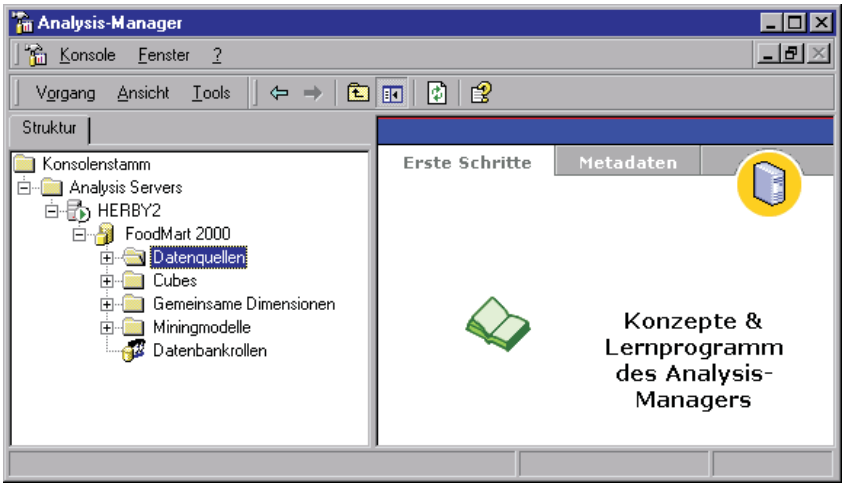


Abbildung 11.6: Der Analysis-Manager

3. Erweitern Sie die Konsolenstruktur, bis DATENQUELLEN erscheint.
4. Klicken Sie mit der rechten Maustaste auf DATENQUELLEN und wählen Sie im Kontextmenü NEUE DATENQUELLE.
5. Markieren Sie für die Access-Tabelle den ODBC-Treiber in der Auswahlliste auf der Registerkarte PROVIDER (siehe Abbildung 11.7).

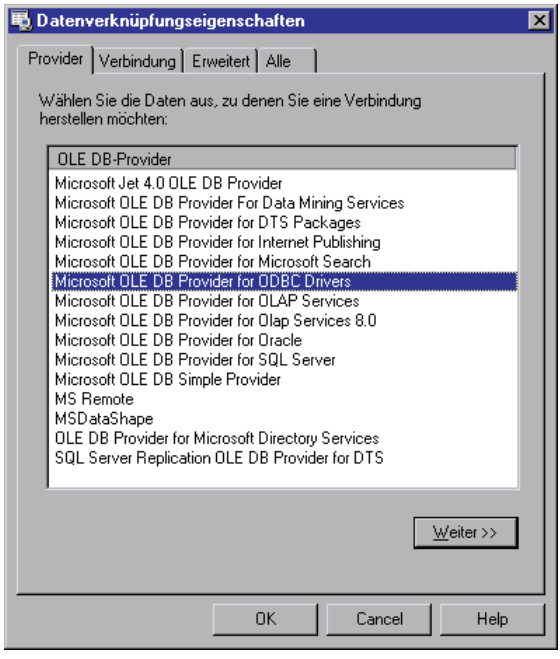


Abbildung 11.7: Auswahl des ODBC-Treibers

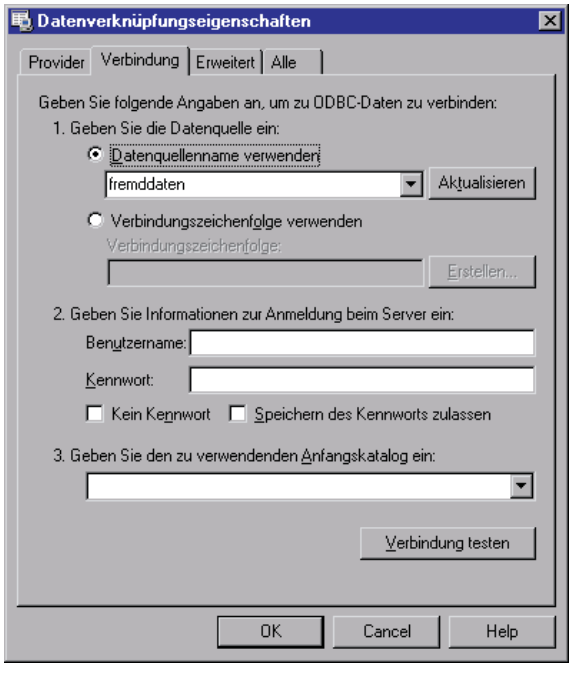


Abbildung 11.8: Die Datenverknüpfungseigenschaften

6. Geben Sie die ODBC-Daten an, die für die Verbindung erforderlich sind (siehe Abbildung 11.7).

7. Klicken Sie auf OK, um die Datenquelle mit dem Analysis Server zu verbinden.

Sie können die ODBC-Verbindung auch noch vor dem Verbinden mit dem *Analysis Server* testen. Für das Einrichten der Beispieldatenbank gilt die gleiche Vorgehensweise. Zusätzlich können weitere Datenquellen hinzugefügt und bestehende bearbeitet werden.

OLAP-Cube erstellen

In einem Cube werden die Daten aus den verschiedenen Quellen zusammengetragen und in eine multidimensionale Struktur gebracht. Für diese Aufgabe verwenden wir den *Cube-Assistenten*.

1. Erweitern Sie im Analysis-Manager die Konsolenstruktur, bis der Eintag CUBES erscheint.

2. Klicken Sie mit der rechten Maustaste auf CUBES und wählen Sie NEUER CUBE bzw. ASSISTENT.

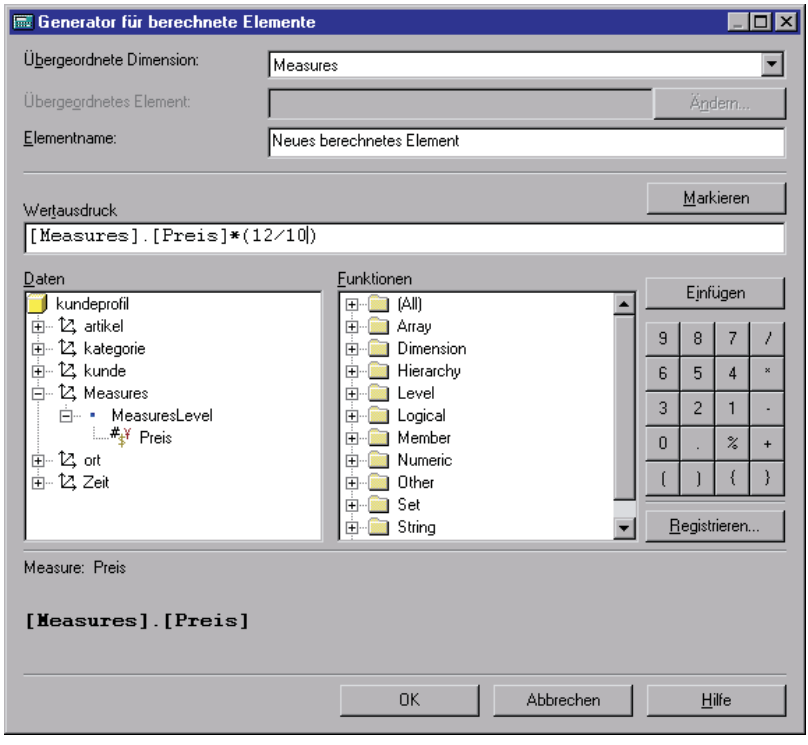


Abbildung 11.9: Auswahl der Faktentabelle

3. Klicken Sie im ersten Dialogfeld auf WEITER.
4. Wählen Sie eine Faktentabelle aus der zuvor gewählten Datenquelle (in unserem Beispiel ist dies die Tabelle mit den Fremddaten aus Excel), wie es in Abbildung 11.9 dargestellt ist.
5. Definieren Sie die Spalte *preis* als Measure.
6. Definieren Sie eine neue Dimension mit dem Dimensions-Assistenten. Klicken Sie dazu auf die Befehlsschaltfläche NEUE DIMENSION.
7. Klicken Sie im Willkommen-Dialogfeld auf WEITER.
8. Erstellen Sie die Dimension im STERNSCHEMA (Standardeinstellung).
9. Wählen Sie als Dimensionstabelle dieselbe Tabelle, die Sie unter Punkt 4 angegeben haben.
10. Geben Sie als Dimensionstypen ZEITDIMENSION und als Datenspalte *bestelldatum* an.
11. Übernehmen Sie die Standardeinstellungen für Zeitdimensionsebene (siehe Abbildung 11.10).

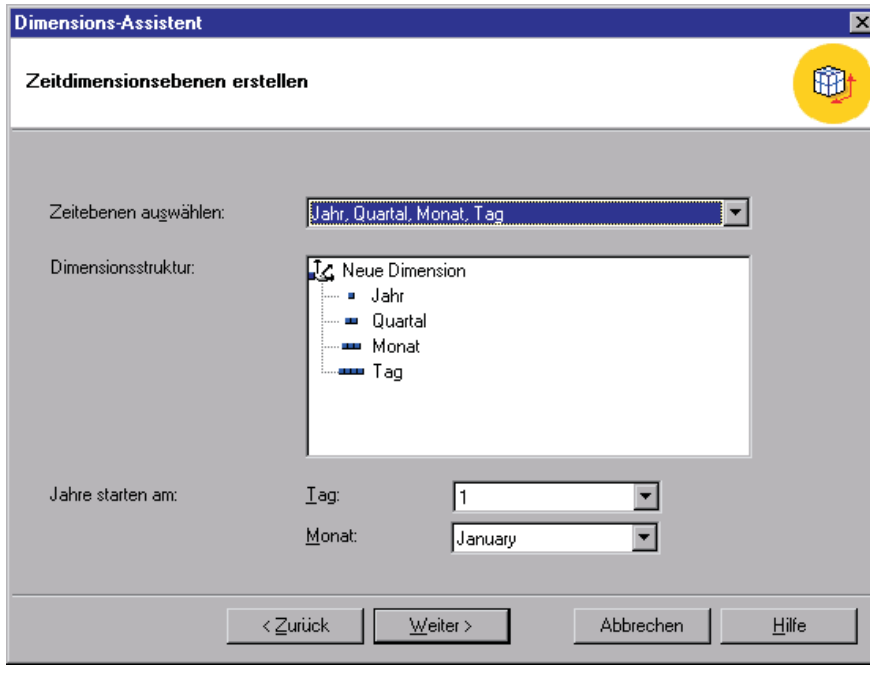


Abbildung 11.10: Erstellen der Zeitdimensionsebene

12. Bei den erweiterten Informationen wählen Sie veränderliche Dimension.
13. Wählen Sie im nächsten Dialogfeld eine Einstellung zur Veränderlichkeit der Dimension.
14. Geben Sie der Dimension einen Namen und arbeiten Sie mit dem Cube-Assistenten weiter.
15. Das System zählt die Zeilen in der Faktentabelle, wenn Sie im nächsten Dialogfeld auf JA klicken.
16. Geben Sie auch einen Namen für den Cube an.
17. Klicken Sie auf FERTIGSTELLEN und der Cube-Editor wird geöffnet.

OLAP-Cube nachbearbeiten

Bevor der Cube-Editor geschlossen und der Cube gespeichert wird, sollen ihm noch weitere Dimensionen und Tabellen zugefügt werden.

1. Klicken Sie mit der rechten Maustaste in den rechten Bildbereich des Cube-Editors.
2. Wählen Sie im Kontextmenü TABELLEN EINFÜGEN.
3. Wählen Sie die Tabelle *adresse* und klicken Sie auf HINZUFÜGEN (siehe Abbildung 11.11).

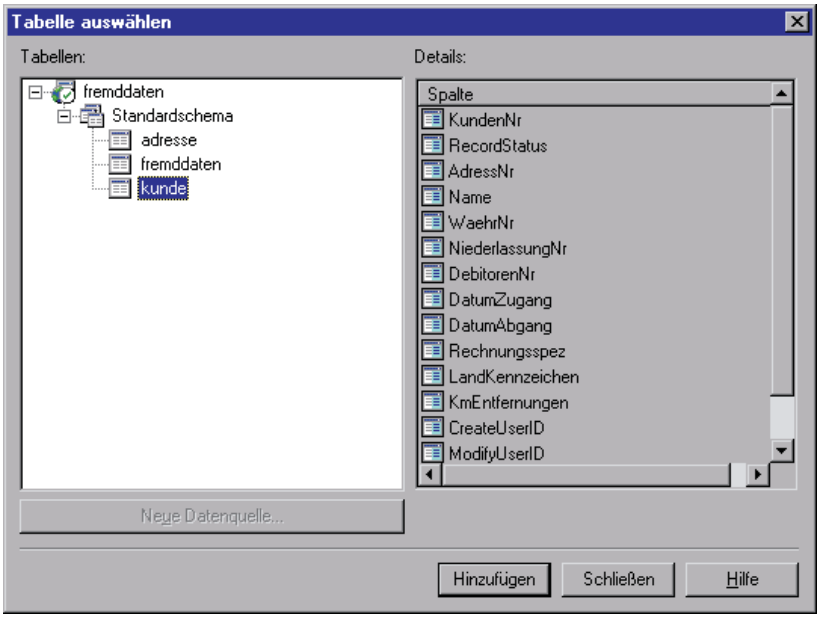


Abbildung 11.11: Auswahl einer Tabelle für den OLAP-Cube

4. Fügen Sie ebenso die Tabelle *kunde* hinzu.
5. Ordnen Sie die Tabellen im rechten Fenster an.
6. Verknüpfen Sie die Tabellen über die Schlüsselspalten per DRAG und DROP.

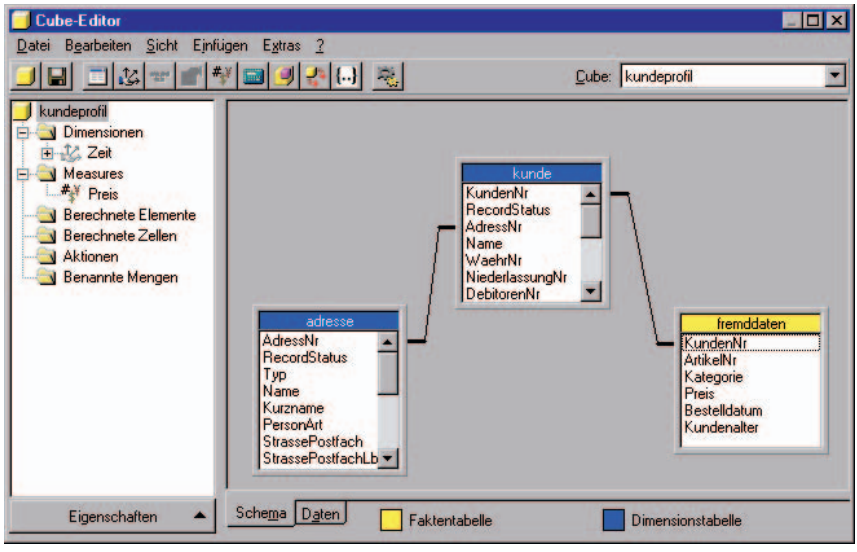


Abbildung 11.12: Verknüpfte Cube-Tabellen

7. Fügen Sie neue Dimensionen hinzu, EINFÜGEN/DIMENSION/NEU...
8. Mit dem Dimensions-Assistenten erstellen Sie die Dimensionen *ort* und *kategorie* mit dem Dimensionstypen *Standarddimension*.

Speicherentwurf für den OLAP-Cube

Nachdem der Cube fertiggestellt wurde, werden die Daten zusammengefasst und berechnet. Dadurch wird eine spätere Abfrage auf den Cube beschleunigt.

1. Schließen Sie den Cube-Assistenten.
2. Wählen Sie MOLAP als Speichertyp.
3. Als Aggregatoption geben Sie »Leistungsgewinn erreicht 50%« an.
4. Starten Sie die Aggregation durch Klicken auf START.
5. Klicken Sie auf WEITER.
6. Lassen Sie die Daten aufbereiten und schließen Sie den Speicherentwurfs-Assistenten.

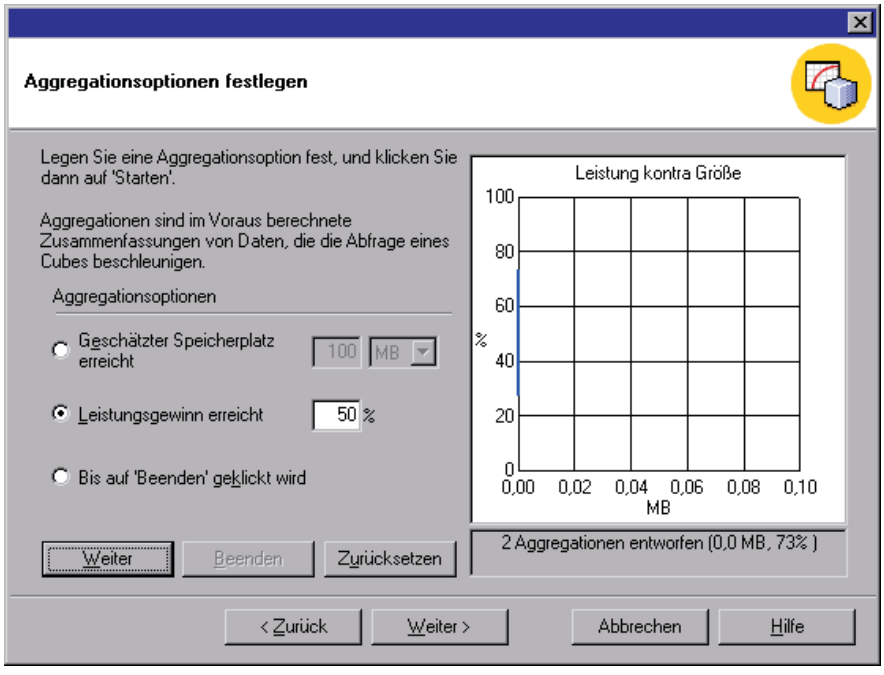


Abbildung 11.13: Die Aggregatsoptionen

Datenanalyse im Cubebrowser

Wurde der Cube mit den zugehörigen Dimensionen erstellt und die Daten aufbereitet, kann man mit der Datenanalyse im Cubebrowser beginnen.

1. Klicken Sie mit der rechten Maustaste im *Analysis Manager* auf den erstellten Cube.
2. Wählen Sie im Kontextmenü DATEN ANZEIGEN.
3. Im Browser sind zunächst alle möglichen Werte für die Dimensionen zugelassen. Der Umsatz ist in der Spalte Preis für das gesamte Jahr zusammengefasst. Wie in Bild 11.14 zu sehen, haben wir eine maximale Verdichtung.
4. Lassen Sie sich den Umsatz je Quartal anzeigen, indem Sie per Doppelklick auf die Zelle »+ 2002« die Zeitangaben aufschlüsseln. Das Ergebnis sollte dann der Abbildung 11.15 entsprechen.
5. Sie können die Verdichtung der Zeit bis zur Tagesangabe aufheben. So können Sie ganz gezielt nach Tagesumsätzen recherchieren (siehe Abbildung 11.16).
6. Bringen Sie die Ergebnistabelle wieder in den ursprünglichen Zustand durch einen Doppelklick auf die Zelle »- Jahr«.

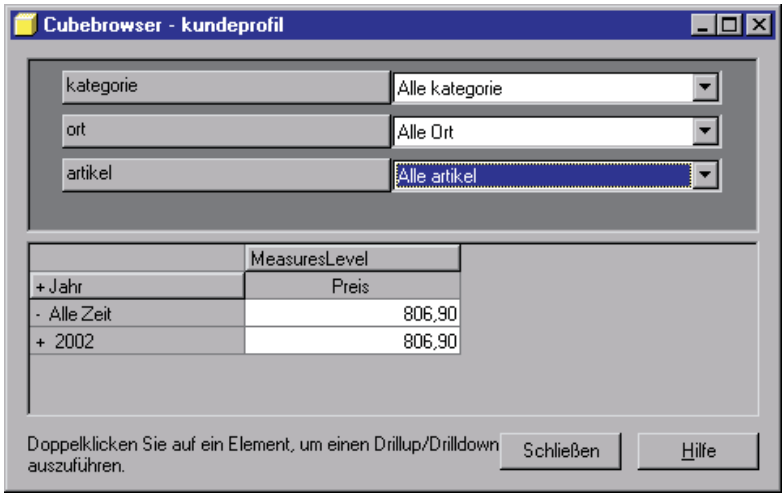


Abbildung 11.14: Der Cubebrowser mit dem Kundenprofil

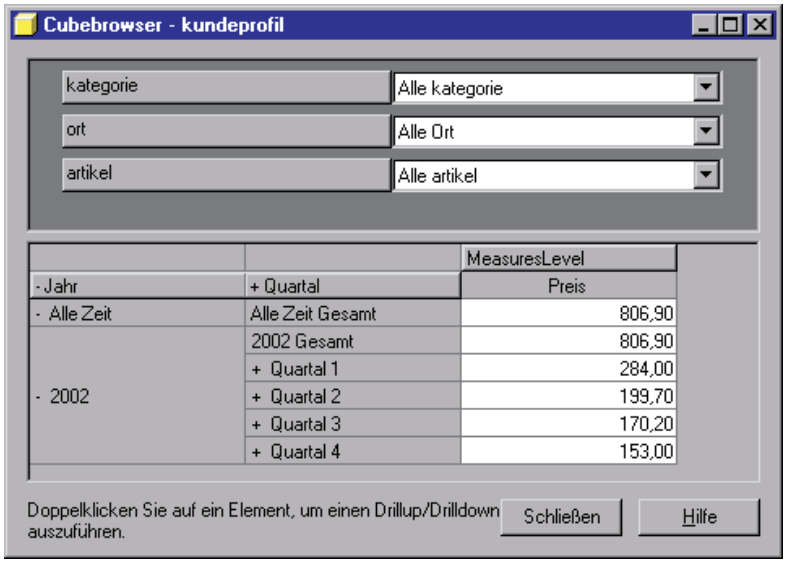


Abbildung 11.15: Kundenprofil mit Quartalsangaben

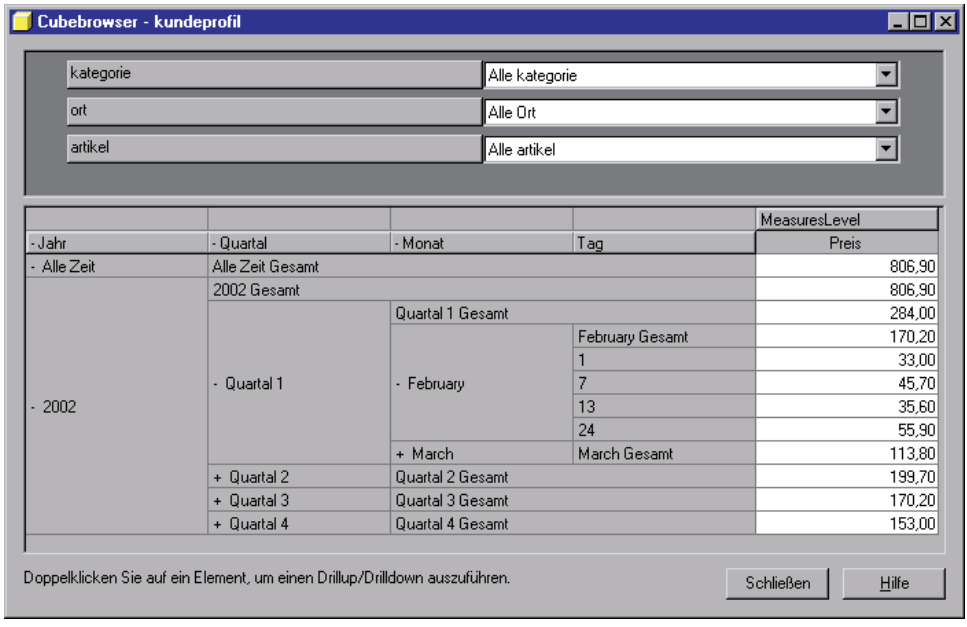


Abbildung 11.16: Der Cubebrowser mit Tagesumsätzen

7. Versuchen Sie nun festzustellen, welche Artikelkategorie sich im Jahr 2002 am besten verkauft hat, indem Sie für die Dimension *Kategorie* die verschiedenen Werte aus der Kombinationsliste auswählen. Das Ergebnis sollte wie in Bild 11.17 aussehen:

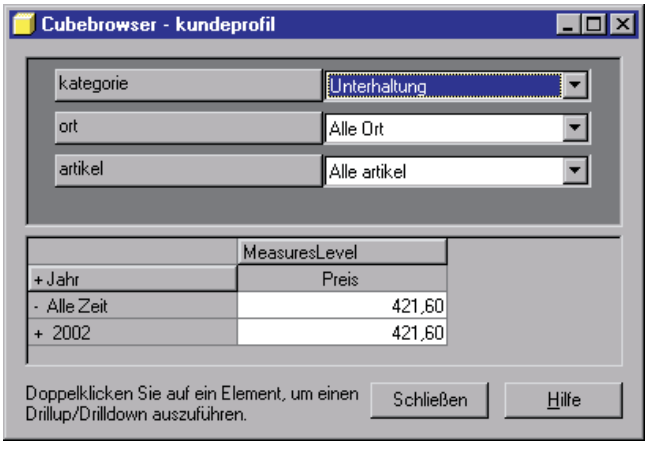


Abbildung 11.17: Kundenprofil nach Kategorie

8. Statt der Zeitachse können Sie sich auch eine Artikelachse für die Ergebnistabelle definieren. Klicken Sie dazu auf die Dimension *artikel* und ziehen Sie den Cursor mit der Maus auf die Zelle »+ Jahr«. Sie werden dann eine Ansicht wie in Abbildung 11.18 haben. In der Ansicht haben Sie jeweils die Preise für die einzelnen Artikel.

9. Schließen Sie den Cubebrowser.

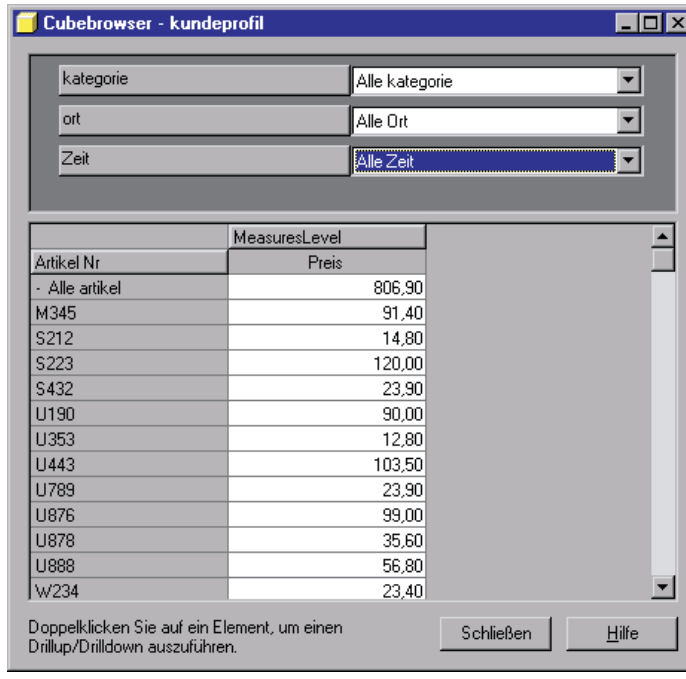


Abbildung 11.18: Kundenprofil je Artikel

Nun haben Sie vielleicht ein Gefühl dafür bekommen, wie Sie im Cubebrowser navigieren können und sehr schnell zu neu definierten Ergebnismengen kommen. Sicherlich haben Sie festgestellt, dass diese Methode effektiver ist, als die Erstellung von SQL-Abfragen und ihre Ausführung. Eventuell haben Sie sich auch gefragt wie Sie mit den gegebenen Dimensionen ein Kundenprofil erstellen sollen. Das war schließlich die Vorgabe, die anfangs definiert wurde. Hierfür ist eine zusätzliche Dimension erforderlich, nämlich die Dimension *kunden*.

Nachträgliches Bearbeiten eines Cubes

1. Öffnen Sie den Cube-Editor, indem Sie mit der rechten Maustaste auf den zu bearbeitenden Cube klicken und im Kontextmenü BEARBEITEN auswählen.
2. Fügen Sie eine neue Dimension hinzu: EINFÜGEN\DIMENSION\NEU...
3. Arbeiten Sie mit dem Dimensions-Assistenten, um die Dimension *kunde* zu erstellen.
4. Außerdem soll neben dem Euro-Preis auch der Dollarpreis ausgewiesen werden. Dafür muss ein berechnetes Element erstellt werden: EINFÜGEN\BERECHNETES ELEMENT...
5. Geben Sie für Dimension, Element und Wertausdruck die Einträge wie in Abbildung 11.19 an.

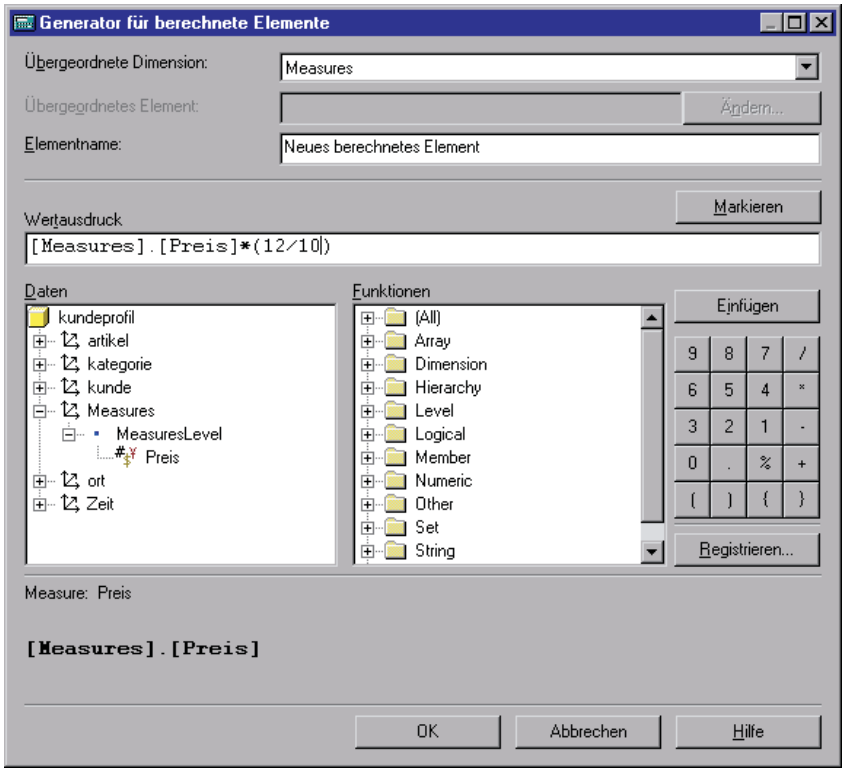


Abbildung 11.19: Nachträgliches Bearbeiten eines Cubes

6. Den Werteausdruck können Sie über die Tastatur oder per Mausklick auf die Daten, Funktionen und Ziffern im Elementegenerator eingeben.

7. Klicken Sie auf OK, um das Element zu erstellen.
8. Im Cubebrowser können Sie sich das Ergebnis anschauen (siehe Abbildung 11.20).

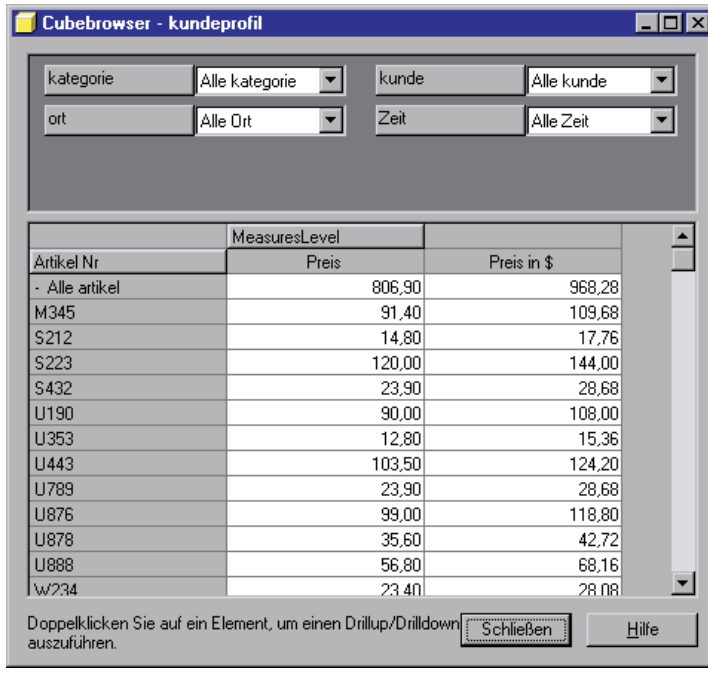


Abbildung 11.20: Überprüfen des Ergebnises im Cubebrowser

Neben den gezeigten Funktionen haben Sie weitere Eingriffsmöglichkeiten, um Ihre Analyse auf ganz konkrete Anforderungen anzupassen. Die Aggregatfunktionen, die Sie bereits von SQL kennen, können auch hier im Zusammenhang mit berechneten Zellen oder Elementen verwendet werden. Darüber hinaus gibt es einige statistische Funktionen, mit denen Sie operieren können. Mit benutzerdefinierten Aktionen können weitere benötigte Daten aus externen Datenquellen bereitgestellt werden.

Um die Daten im *Cube* zu schützen sollten Sie auch hier, wie es bereits im Kapitel »Sicherheit und Wartung« beschrieben wurde, Benutzer und Gruppenkonten einrichten und die Zugriffsberechtigungen verwalten.

Die Analyse über den Cubebrowser ist eine und zunächst die einfachste Form, die Daten im Cube auszuwerten. Mit den Protokollen HTTP oder TCP/IP kann eine Verbindung vom Server zu Client-Produkten wie MS Excel hergestellt werden, wo die Daten grafisch aufbereitet werden können.

Mit der Implementation von Dataminingmodellen können in den Cubes bestimmte Kundenmuster erkannt werden. Demografische Informationen über die Kunden und

ihr Kaufverhalten werden gemeinsam analysiert und die *Korrelationen* in Mustern zusammengetragen. Mit diesen Mustern könnte beispielsweise die Marketingabteilung das Profil der angebotenen Produkt- oder Servicepalette verkaufsfördernd an die Kundenbedürfnisse anpassen.

11.2.3 Nachteile beim Arbeiten mit OLAP

Es ist nicht alles Gold was glänzt. Auch bei den OLAP-Datenbanken gibt es, wie kann es bei einer relativ neuen Technologie auch anders sein, Schwierigkeiten. Diese sind auch in der üblicherweise großen Menge an zu verarbeitenden Daten begründet.

- Echtzeit-Datenbestände sind nur begrenzt möglich.
- Kein Standard bei den Abfragemethoden.
- Laufzeiten (beim Laden und bei Kalkulationsvorgängen) lassen sich nur schwierig voraussagen.
- Große Datenvolumen resultieren aus Kalkulationsvorgängen.
- Handhabung einiger OLAP-Produkte ist recht umständlich.

11.3 Publizieren im Web

Durch die Kommerzialisierung des Internets Mitte der 90er Jahre wurde es auch für viele Unternehmen interessant, Angebote im »Netz der Netze« feilzubieten. Schnell ging der Trend hin zu dynamischen Webseiten, die ihre Informationen aus ständig aktualisierten Datenbanken erhalten. Aber nicht nur für Kundeninformationen im Internet ist der Zugriff auf die Datenbank bzw. Teilbereiche der Datenbank denkbar, auch intern ist ein Zugriff über einen Browser im Intranet durchaus sinnvoll. Web-Browser sind plattformunabhängig und können nahezu auf jedem Client-PC installiert werden. Das erspart zusätzliche Kosten für spezielle Software.

11.3.1 Technische Voraussetzungen

Die Minimalvoraussetzungen für eine webbasierte Datenbanklösung sehen wir in Abbildung 11.21. Es reicht funktional aus, ein DBMS und einen *Web-Server* auf einem Rechner zu installieren. Das DBMS liefert die Informationen, die in dem HTML-Dokument aufbereitet werden und vom Client auf einem Web-Browser angesehen werden können.

Der Web-Server, auf dem das HTML-Dokument liegt, und das DBMS kommunizieren über eine Schnittstelle, die wir hier *Internet-DB-Connector (IDC)* nennen. Bei dieser Minimallösung spricht man auch von einer Standalone-Web-Datenbank.

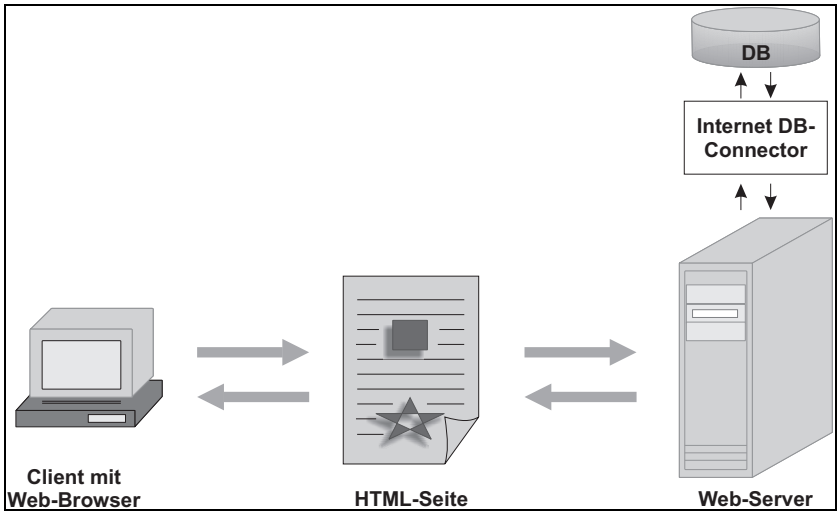


Abbildung 11.21: Internetanbindung mit IDC und Web-Server

Die soeben beschriebene Lösung ist für größere Unternehmen sicherlich nicht geeignet. Dort wird man eine Lösung suchen, bei der ein Web-Server in das bestehende Netzwerk integriert und das bisher genutzte DBMS als Back-End-Datenbank verwendet wird. In diesem Fall wird der Aufwand für die Berücksichtigung der Sicherheitsaspekte unvermeidbar anwachsen. Ein Konzept mit Firewall-Einbindung ist unbedingt erforderlich.

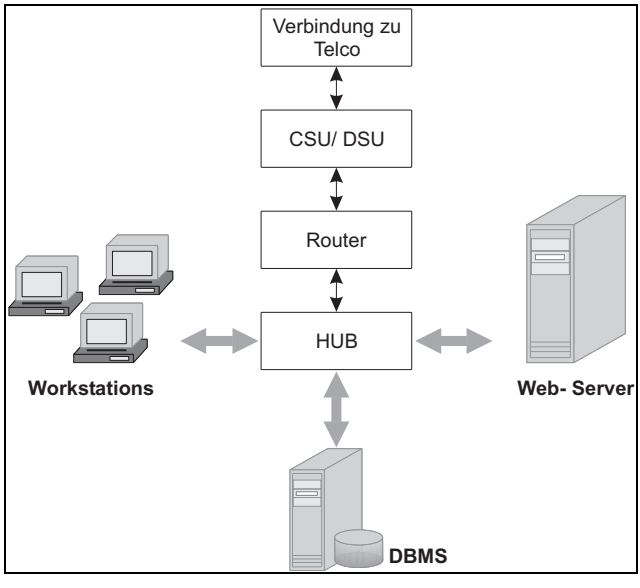


Abbildung 11.22: Technische Komponenten für die Internetanbindung

Mit einem Webserver allein ist die Verbindung zur »großen weiten Welt« noch nicht gegeben. Es gibt verschiedene Internet-Verbindungen, die von den *Internet Service Providern* (ISPs) angeboten werden und in der Tabelle 11.1 vorgestellt werden.

Internetverbindung	Bandbreite	Vorteile	Nachteile
Frame Relay	56 KB – 128 KB	Keine entfernungsabhängigen Kosten.	Ungeeignet für Sprache und andere zeitkritische Dienste.
Point-to-Point	< 155 MB	Hohe Bandbreite.	Hohe Kosten.
ISDN	32KB – 128KB	Verfügbar auf Abruf. Einsetzbar als Überlaufkapazität für Spitzenlasten.	Bandbreite wird auch von Telefon und Fax beansprucht.
SLIP/PPP	56.6 Kbaud	Verbindung bei Bedarf.	Niedrige Bandbreite.
Digital Subscriber Line (DSL)	256 KB – 2300 KB	Hohe Bandbeite bei vergleichsweise geringen Kosten. Keine neue Verkabelung.	Noch nicht überall verfügbar.

Tabelle 11.1: Vor- und Nachteile von Internetverbindungen

Damit das Local Area Network (LAN) an das Internet angeschlossen werden kann, ist neben der direkten Verbindung zum ISP auch zusätzliche Hardware erforderlich. In der Abbildung 11.22 sind die einzelnen Komponenten dargestellt. Router, Hub und Server sind sowieso Basiskomponenten eines Netzwerkes. Eine neue, für die Internet-Anbindung notwendige Hardwarekomponente, ist eine Vorrichtung zur Kommunikation zwischen LAN und ISP. Diese Einheit nennt man auch *Customer Service Unit/Data Service Unit (CSU/DSU)*.F Ähnlich einem Modem werden bei einem CSU/DSU die Daten für den Transfer über eine Datenleitung aufbereitet. Auf der anderen Seite der Leitung muss dann auch wieder eine entsprechende Komponente vorhanden sein, welche die Daten wieder umformt.

11.3.2 Welche Daten gehören ins Internet

Neben den bereits vorgestellten Sicherheitsmaßnahmen (siehe Kapitel 9) stellt sich zunächst die Frage nach der Erfordernis eines Internetanschlusses. Es sollte eine Analyse über den Kommunikations- und Veröffentlichungsbedarf durchgeführt werden. Personenbezogene Daten haben in den seltensten Fällen etwas in Internet zu suchen. Vermeiden Sie es, die Daten von Mitarbeitern und erst recht die der Kunden im Internet verfügbar zu machen. Dazu gehört auch, dass diese Daten einem besonderen Schutz unterliegen.

Eine besondere Freude wird der Besucher ihrer Web-Seiten haben, wenn er einen echten Informationsgewinn daraus ziehen kann, d.h. neben all der Werbung wäre ein wenig Informatives nicht schlecht. Im Folgenden sollen einige Beispiele zeigen, welche Daten im Internet durchaus sinnvoll wären:

- Produktinformationen,
- Fahrpläne,
- Vereinsbeschreibungen,
- Adresslisten von Behörden,
- Online Support,
- Tarifbestimmungen,
- Fernseh- und Rundfunkprogramme,
- Gesetzestexte,
- Nachrichten.

11.3.3 Verfahren zur Anbindung der Datenbank an das Internet

Bei der Auswahl der geeigneten Technologie, welche die Anbindung der Datenbank ans Web ermöglichen soll, hat der Entwickler die Qual der Wahl. Viele Hersteller drängen ins Internet und alle versuchen ihr Produkt so gut wie es eben geht zu positionieren. Dazu gehört auch, dass herstellerspezifische Schnittstellen entwickelt werden, die andere Anbieter vom Markt drängen soll. Neben der bestmöglichen Perfomance stellt sich also auch die Frage, mit welcher Soft- und Hardware Sie bisher ausgerüstet sind und welche Technologie zur Datenbankanbindung die bestehenden Komponenten am besten ergänzt.

Hypertext Transfer Protocol (HTTP)

HTTP ist ein Protokoll, welches die Kommunikation zwischen HTTP-fähigen Servern und Clients ermöglicht. Die dadurch stattfindende plattformunabhängige Kommunikation hat dem Internet erst die Verbreitungschance geschaffen. Was als Forschungsprojekt des europäischen *Teilchenforschungszentrums für den elektronischen Informationsaustausch* begann, entwickelte sich schnell zu einem globalen Informationssystem. HTTP ist speziell auf das Navigieren in Hypertexten ausgerichtet und verzichtet auf das Erzeugen einer dauerhaften Verbindung zwischen den kommunizierenden Rechnern. Es werden einzelne Fragen und Antworten verschickt und die Verbindung wird nach Erhalt einer Antwort wieder unterbrochen. Der Server stellt dem Client Hypermedia-Dokumente zur Verfügung. Ein Dokument kann vom Browser eines Clients über ein *Uniform Resource Locator (URL)*, also über eine eindeutige Bezeichnung, aufgerufen werden. Die Syntax eines solchen Aufrufes ist jedem geläufig, der im Internet ein Dokument aufgerufen hat:

http://www.addison-wesley.de

Diese eindeutigen Bezeichner, sind nun in Hypermedia-Dokumenten eingebettet, wodurch eine Verbindung zu dem entsprechenden Dokument erstellt und die alte Verbindung unterbrochen werden kann.

Ein weiterer großer Vorteil von HTTP ist die Fähigkeit, Kommunikation zwischen Rechnern auch mit anderen Protokollen zu ermöglichen. HTTP stellt dabei die Basis dar, auf der weitere Internetdienste wie SMTP Mail, FTP oder Gopher ansetzen können. Außerdem können verschiedene Datentypen übertragen werden. Diese Eigenschaft von HTTP hat zur Folge, dass auf Hypermedia-Dokumenten auch Bilder und Videosequenzen dargestellt werden können.

Mit HTTP können Sie auch direkt auf eine Datenbank zugreifen. Das Beispiel 11.1 zeigt, wie Sie mit einer URL eine Menge aus der Datenbank *sample* abfragen und diese an ein XML-Dokument weiterleiten.

Beispiel 11.1:

http://IISServer/vsam?sql=SELECT++FROM+service+FOR+XML+AUTO&root=root*

Der *root-Parameter* identifiziert ein einzelnes Element der obersten Ebene. Das virtuelle Verzeichnis *vsam* muss bereits erstellt worden sein.

Common Gateway Interface (CGI)

Eine Schnittstelle für den Datenaustausch haben wir beim *Common Gateway Interface* (CGI). Man kann bei CGI auch von einem Programm sprechen, das eine Verbindung zwischen einem Information-Server und einer externen Anwendung herstellt. Ein Web-Server kann z.B. mittels CGI Daten mit dem DBMS austauschen.

Ein CGI-Skript ist eine Software, die auf dem Server ausgeführt wird, wenn ein Client seine URL aufruft, die mit dem entsprechenden *Gateway* korrespondiert. Das Ausführungsergebnis wird dann direkt an den Client weitergeleitet.

CGI-Skripte werden von vielen Entwicklungsumgebungen unterstützt (z.B. Visual Basic, C/C++, Java etc.).

In der Abbildung 11.23 sehen Sie, wie ein HTTP-Server mit einer Datenbank per CGI und API kommunizieren kann.

Ein weitere interessante Lösung zur Herstellung einer Web-Datenbankverbindung ist der Gebrauch von CGI-basierten Entwicklungsprodukten. Die Handhabung dieser Produkte ist recht einfach. Es handelt sich um eine Kombination aus Standard-HTML und einer eigenen Datenbank-Beschreibungssprache. Die Abbildung 11.24 zeigt eine solche Lösung. Die *Cold Fusion Engine* ist ein CGI-Datenbank-Skript der Firma Allaire.

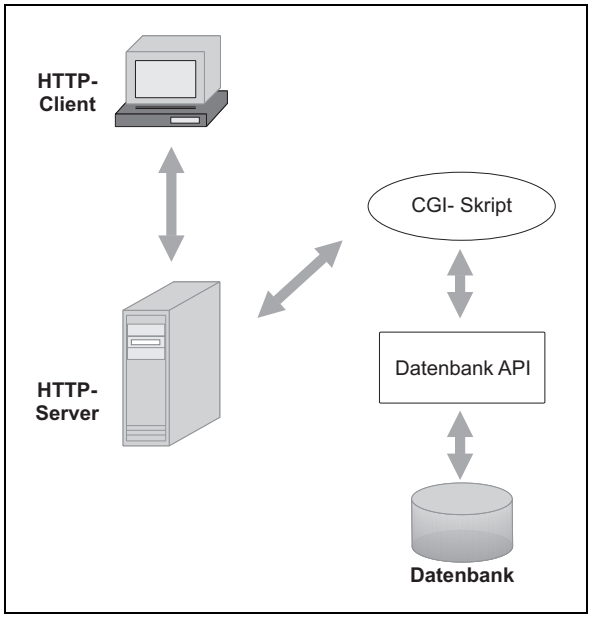


Abbildung 11.23: Architektur mit CGI

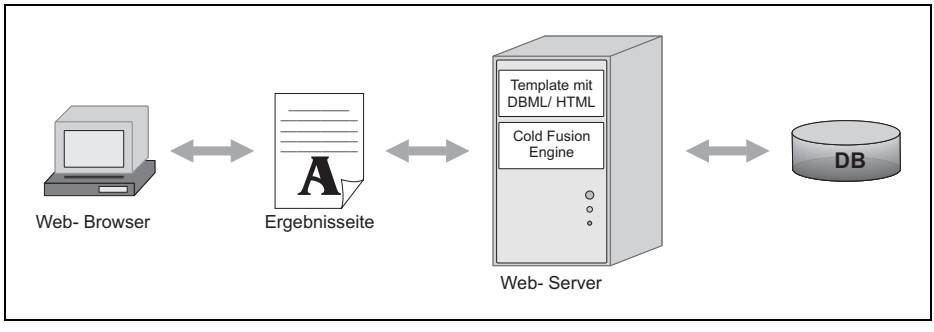


Abbildung 11.24: Web-Server mit Cold Fusion

Wird vom Client eine Abfrage über eine HTML-Seite an den Server geschickt, so wird auf dem Server zunächst die DBML-Befehle des Skriptes ausgeführt. Die Cold Fusion Engine wertet die Auswahlkriterien des Clients aus und stellt eine Verbindung (meist über ODBC) mit der Datenbank her. Cold Fusion erzeugt mit der Ergebnismenge aus der Datenbank eine dynamische HTML-Seite, die der Server an den Client weiterleitet.

Hypertext Preprozessor (PHP)

PHP, das ursprünglich von Rasmus Lerdorf entwickelt wurde, zählt zu den Server-Skriptsprachen, deren Skripte sich in »normale« HTML-Dateien integrieren lassen. Bei PHP werden die in die HTML-Dateien integrierten Server-Skripte ausgewertet und als normales HTML an den Client ausgegeben. Der Browser »sieht« also nichts von dem PHP-Code. Eine Stärke von PHP ist die Unterstützung sehr vieler DBMS, zu denen Informix, mSQL, MySQL, MS SQL Server, Oracle, PostgreSQL und Sybase gehören. ODBC-Zugriffsfunktionen bietet PHP ebenfalls an, so dass alle Datenbanksysteme mit ODBC-Treiber hinzukommen. Eines dieser Datenbanksysteme mit ODBC-Treiber ist MS Access. Ein Vorteil von PHP gegenüber ASP ist die Tatsache, dass PHP alle relevanten Plattformen unterstützt. Zu den Plattformen, die von PHP unterstützt werden, zählen auch die Betriebssysteme Linux und Windows NT.

Internet Database Connector (IDC)

Der *Internet Database Connector* ist eine Server-Erweiterung des Internet Information Server ab Version 1.0. Diese Erweiterung ermöglicht den Zugriff auf Datenbanken über die Datenbankschnittstelle ODBC. Der Internet Database Connector ist als Dynamic Link Library (DLL) der Server-Schnittstelle ISAPI implementiert.

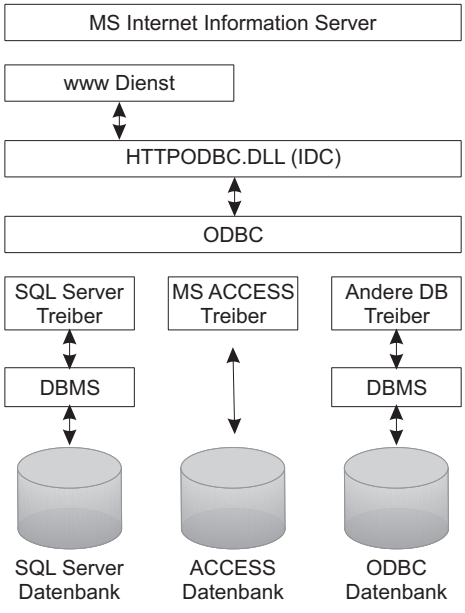


Abbildung 11.25: Die Komponenten für eine Architektur mit ODBC, IIS und IDC

Für den Zugriff auf Datenbanken über IDC werden immer zwei Dateitypen benötigt. Die so genannte *IDC-Datei* mit der Endung .IDC enthält die Datenbankabfrage als SQL-Statement und den Namen der ODBC-Datenquelle. Die HTX-Datei (HTX = HTML Extension) mit der Endung .HTX ist als HTML-Schablone zu verstehen, in welche die Daten aus der Datenbank eingetragen werden. Somit entsteht für den Browser eine ganz normale HTML-Datei.

Die Abbildung 11.25 zeigt, wie ein Internet Information Server mit DBMS zusammenarbeiten kann. Die IDC-Datei enthält die notwendigen Informationen, um eine Verbindung mit der angeforderten ODBS-Datenquelle aufzubauen und eine SQL-Anweisung auszuführen. Die IDC-Datei ist eine formatierte Textdatei auf einem IIS-Webserver, die drei Informationsbereiche enthält: eine ODBC-Datenquelle, den Verweis zu einer Template-Datei (HTX-Datei) und die SQL-Anweisung.

Active Server Pages (ASP)

ASP ist eine Server-Erweiterung des Internet Information Server 4.0, welche die Ausführung von Skripten auf Server-Seite erlaubt. ASP eine optimierte Objekt- oder Komponentenbibliothek. ASP selbst ist keine Skriptsprache, sondern eine Umgebung, welche die Ausführung von Skripten auf Server-Seite ermöglicht. Standardmäßig unterstützt ASP die Skriptsprachen VBScript, welches Visual Basic sehr ähnlich ist, und JScript, das sich an JavaScript von Netscape orientiert.

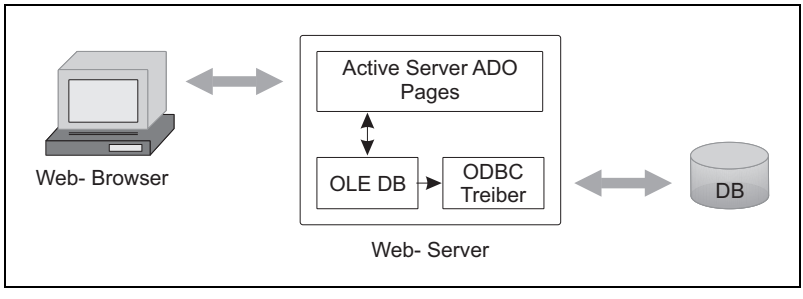


Abbildung 11.26: Die Servererweiterung ASP

Die ASP-Umgebung ist allerdings nicht auf diese zwei Skriptsprachen festgelegt. Es können auch andere Skriptsprachen installiert werden.

ASP-Skripte werden, wie bei PHP, in HTML-Seiten integriert. Die Skripte werden durch die Kennzeichen <% und %> von den HTML-Elementen getrennt.

Die Abbildung 11.26 zeigt die Architektur einer Datenbankanbindung mit ASP. Ein ActiveX-Datenobjekt (ADO) befindet sich innerhalb einer ASP-Datei auf dem Web-Server. Wird vom Browser eine Abfrage an den Web-Server gesendet, so wird die ASP-

Datei aufgerufen und von dort wird die Anfrage via OLE DB an die Datenbank weitergeleitet. ASP setzt die Ergebnismenge entsprechend um und über HTTP kann der Client das Ergebnis auf seinem Browser sehen.

11.3.4 Datenbeschreibungssprache XML

Immer wieder gibt es Technologien, die ein gewaltiges Medieninteresse hervorrufen und die gesamte Branche aufmischen. Die Datenbeschreibungssprache XML gehört zu diesen, mit viel Hoffnungen getränkten, Schlagwörtern. Von Evolution, ja sogar von Revolution war die Rede. Tatsächlich stellt der neue Standard in mehreren Bereichen einen echten Fortschritt dar und hat sich in kurzer Zeit zum Standard beim Datenaustausch etabliert. Warum XML auch die Datenbanktechnologie betrifft und deshalb Berücksichtigung in diesem Buch findet, soll im weiteren Verlauf beschrieben werden.

XML steht für *Extensible Markup Language*, ins Deutsche übersetzt heißt dies soviel wie »Erweiterbare Auszeichnungs-Sprache«. Ist XML also eine Weiterentwicklung und ein Ersatz für die Beschreibungssprache HTML (Hypertext Markup Language)? HTML ist im Internet bisher für die Erstellung, Übertragung und Vernetzung von Dokumenten zuständig. Die Funktion von HTML und XML sind grundsätzlich voneinander abweichend. HTML wurde entwickelt, um Daten anzuzeigen und ihr »Aussehen« zu beschreiben. Im Gegensatz dazu werden durch XML die Daten als solches beschrieben. HTML stellt die Information dar und XML beschreibt die Information. XML manipuliert die Daten also nicht, sondern ist schlicht und einfach eine Möglichkeit, die Struktur der Daten zu beschreiben. Das erscheint bei all dem Wirbel, der um diesen Begriff gemacht wurde, vielleicht etwas ernüchternd. Dennoch ist XML insbesondere im Datenbankbereich sehr vielseitig einsetzbar.

Im Beispiel 11.2 werden Dienstleistungsdaten mit XML beschrieben. Ebenso wie HTML verwendet XML Auszeichnungsmerkmale, die hier allerdings nicht vordefiniert sind, sondern vom Autor des XML-Dokumentes selbst entworfen werden können.

Beispiel 11.2:

```
<service>
<bezeichnung> SUPT23 </ bezeichnung >
<beschreibung>Telefonsupport </beschreibung>
<preis waehrung = „Euro“> 75,-</preis>
</service>
```

Die Merkmale zur Auszeichnung von Dokumenten werden auch *Tags* genannt. So ist ein variablere und präzisere Beschreibung der Information möglich. Die Tags werden eingesetzt, um Strukturelemente eines Dokuments bedeutungsorientiert (semantisch) zu beschreiben. Und sie können variabel an die Bedürfnisse einer Anwendung angepasst werden (siehe Abbildung 11.27).

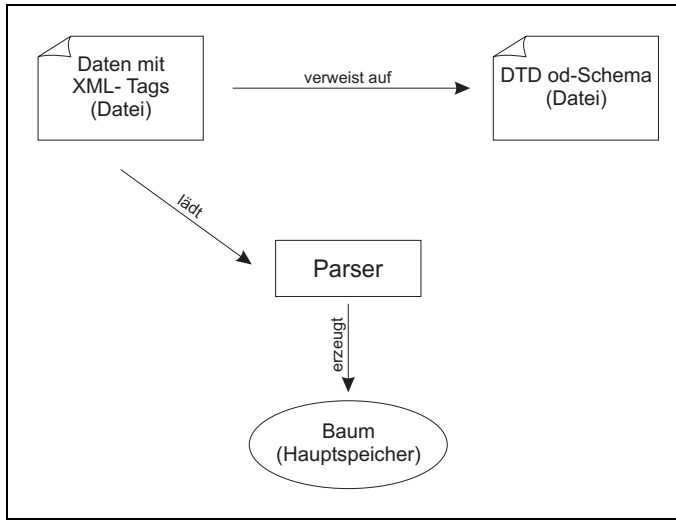


Abbildung 11.27: Verarbeitung von XML-Dateien

Vom XML-Dokument gibt es einen Verweis auf eine DTD- (Document Type Definitions) oder Schemadatei, welche die Struktur im XML-Dokument beschreibt. Wird ein XML Dokument geschrieben, so kann es auf die passende DTD (oder Schema) zugreifen, damit der Datenbaum richtig interpretiert werden kann. In der DTD stehen die einzelnen möglichen Tags und Attribute für das XML-Dokument. Die Reihenfolge der Vereinbarungen ist nicht entscheidend, sie muss jedoch vollständig sein. Die DTD gibt genau die Baumstruktur der XML-Seite wieder. Damit das Erstellen der DTD gelingt, muss die Datenstruktur der XML-Dokumente klar definiert sein.

Es gibt bereits DBMS, die XML-Dokumente aufnehmen und verarbeiten. Die einfachste Form ist, XML-Dokumente in einem CLOB (siehe Kapitel 12) unterzubringen. Die Tags und der Dokumentinhalt werden dann wie Fließtext behandelt und mit den Methoden der Volltextsuche kann man sich den Inhalt der abgespeicherten Dokumente erschließen. Es gibt aber auch DBMS, die XML-Strukturen sehr viel direkter unterstützen. Doch der XML-Support in Datenbanken könnte nach wie vor besser sein.

Ein weiterer Einsatzbereich von XML liegt im Informationsaustausch zwischen heterogenen Datenbanksystemen (siehe Abbildung 11.28).

Zwei Geschäftspartner müssen miteinander kommunizieren und dabei zum Beispiel Produktdaten austauschen. Beide Partner haben jedoch ganz unterschiedliche HW-Plattformen und Datenbanksysteme im Einsatz. Die Geschäftspartner haben ihre jeweils eigene »Begriffswelt«. XML stellt für solche Problemstellungen ein ideales *Datenaustauschformat* zur Verfügung. Die Geschäftspartner einigen sich auf eine Darstellung, also eine Dokumenttypdefinition, und stellen Konverter zur Verfügung, die die übertragenen Dokumente in die jeweiligen *Inhouse-Systeme* umsetzt.

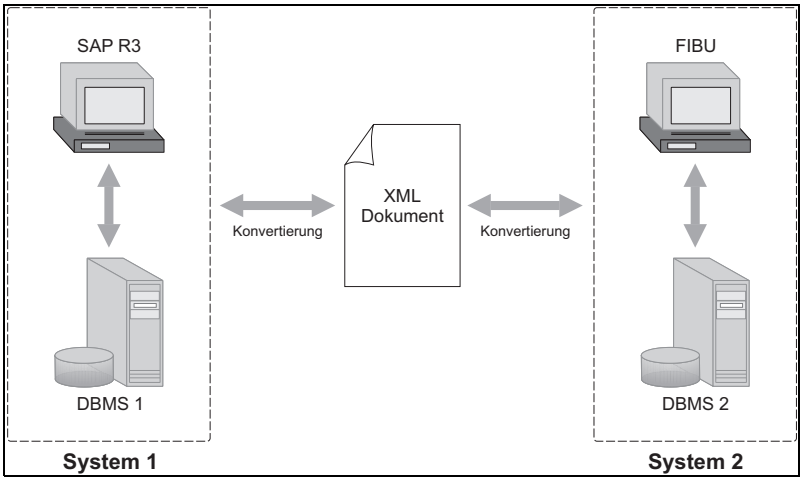


Abbildung 11.28: Datenaustausch mit XML

Hinter diesem Beispiel verbirgt sich ein bedeutendes Rationalisierungspotential. Laut einer Untersuchung investieren Unternehmen bis zu 40% ihres IT-Budgets, um den Datenaustausch zwischen verschiedenen Datenbanksystemen zu ermöglichen oder zu optimieren. Laut Meinung der Experten heißt die Lösung XML.

Der neue Internet-Standard ist sehr flexibel einsetzbar. Man kann sich zum Beispiel auch sehr gut vorstellen, relationale oder objektrelationale Strukturen mit XML zu beschreiben. Wir setzen XML-Dokumente dann nicht mehr als Objekte in relationalen oder objektrelationalen Datenbanken ein, sondern wir machen die Datenbankanwendungen zu XML-Dokumenten.

Zusammenfassend soll nochmals gezeigt werden, wo die Stärken von XML liegen:

- XML ermöglicht eine Trennung von Struktur und Formatierung der Daten.
- XML kann zum Datenaustausch in heterogenen Systemen genutzt werden.
- Informationsaustausch über das Internet (Business to Business).
- XML beschreibt, wie Daten in Dateien oder Datenbanken gespeichert werden.
- XML kann von verschiedenen Benutzern ausgewertet werden (nicht nur über Browser).

11.3.5 Unternehmensdaten im Web (Beispieldatenbank)

Hat man sich dazu entschlossen, die Datenbank ans Web anzubinden und für die gegebenen Voraussetzungen die beste Methode gewählt, so ist es an der Zeit, die notwendigen Etappen zum Ziel zu beschreiten. Ein Teilbereich der Unternehmensdatenbank soll im Internet abgebildet werden. Der Servicebereich, der in der Beispieldatenbank durch

die Tabellen *service*, *kategorie* und *bilder* repräsentiert wird, soll den Kunden via Web vorgestellt werden. Eine einfache Möglichkeit, die in einem folgenden Abschnitt auch beschrieben werden soll, wäre, die Daten durch einen HTML-Export des DBMS einmalig aufzubereiten und die HTML-Seiten auf den Web-Server zu transferieren. Man nennt diese Web-Seiten auch statischen Web-Seiten, deren Inhalte gleichbleibend sind und nicht durch Interaktivität verändert werden können. Eine Alternative dazu sind dynamische Web-Seiten, die in Abhängigkeit der bestimmten Auswahlkriterien den Inhalt der Seite aus der Datenbank speist. Die Anbindung an die Datenbank wird anhand des Zusammenspiels vom Internet Information Server, dem Protokoll HTTP und den Internet Database Connector vorgestellt. Es sei nochmals darauf hingewiesen, dass die hier gezeigte Möglichkeit eine von vielen ist.

Statische HTML-Seite

Zunächst soll der einfache Lösungsansatz beschrieben werden, wie Sie die Exportfunktionalität ihres DBMS nutzen können, um aus den Informationen Ihrer Datenbank eine statische HTML-Seite zu erzeugen. Diese Seiten mögen Ihnen vielleicht etwas »trocken« erscheinen, aber hat das Unternehmen keinen eigenen Web-Server, dann ist sie darauf angewiesen ihrem Internet Service Provider per FTP statische Seiten zu schicken. Für die Vorstellung der Unternehmensziele und -angebote können Internetseiten mit statischem Inhalt auch völlig ausreichen.

Mit der Web-Funktionalität von MS Access können Sie sehr einfach und komfortabel die Informationen aus den Tabellen in HTML exportieren. Wir wollen mit unserer Arbeit dort beginnen, wo wir im Kapitel »Berichtserstellung« aufgehört haben. Die Grundlage für das HTML-Dokument sei ein erstellter Bericht. Wenn Sie den Bericht in Kapitel 10 nicht selbst erstellt haben, können Sie auch die Access-Datei BEISPIELDB.MDB auf der Begleit-CD verwenden. Die weiteren Aktionen vom Bericht zum fertigen HTML-Dokument sind folgende:

1. Starten Sie MS Access und öffnen Sie die Beispieldatenbank.
2. Wählen Sie die Datenbankobjekte BERICHTE.
3. Es erscheint die Liste mit den vorhandenen Berichten in der Datenbank. Klicken Sie mit der rechten Maustaste auf den Bericht RPTSERVICES.
4. Wählen Sie im Kontextmenü EXPORTIEREN...
5. Geben Sie im Export-Dialogfeld HTML-Dokumente bei Dateityp an. Vergeben Sie einen Dateinamen und das Verzeichnis, in das die HTML-Datei exportiert werden soll (siehe Abbildung 11.29).
6. Klicken Sie auf SPEICHERN.
7. Wählen Sie eine HTML-Vorlage.
8. Das HTML-Dokument wurde nun erstellt und Sie können es mit einem Web-Browser anschauen.

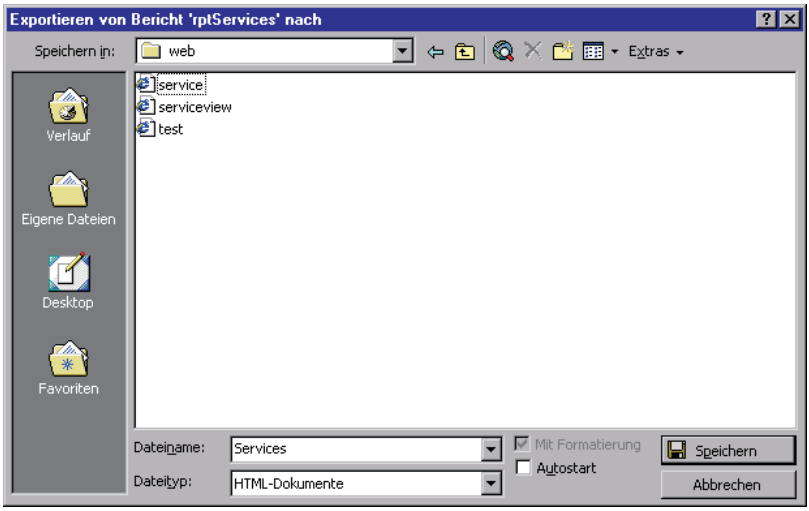


Abbildung 11.29: HTML-Export in MS Access

Das Ergebnis, wie auf Abbildung 11.29 gezeigt, lässt sicherlich noch zu wünschen übrig, aber Sie haben Ihre ersten Daten in ein HTML-Dokument übertragen. Wenn das Exportergebnis nicht Ihren Vorstellungen entspricht, können Sie es auch mit einem HTML-Editor nachbearbeiten.

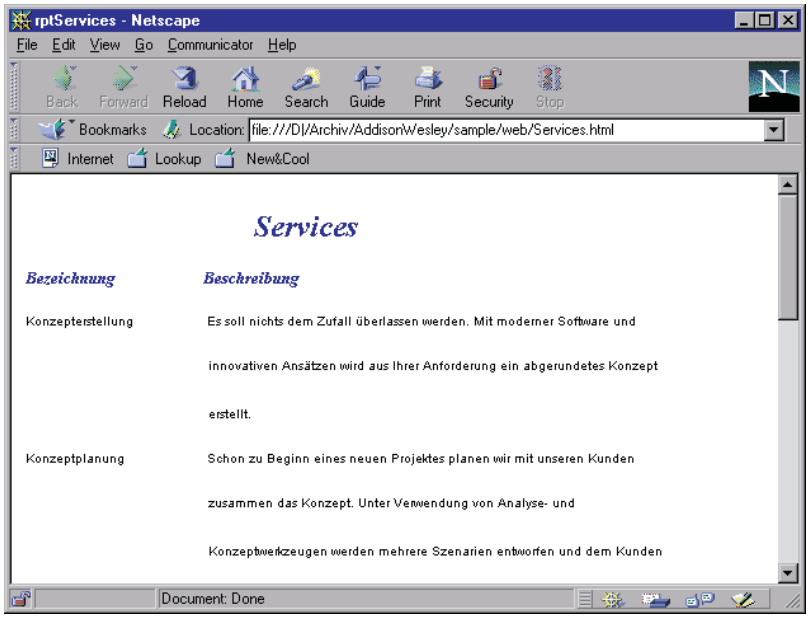


Abbildung 11.30: Exportierte HTML-Seite im Browser geöffnet

Dynamische HTML-Seiten

In diesem Abschnitt lernen Sie, wie Sie Ihre Datenbank an das Internet oder Intranet anbinden. Das gewählte Verfahren für die Anbindung sei eine Verbindung über *IDC*. Dies Verfahren wurde gewählt, weil es mit wenig Aufwand eine Datenbankabfrage über einen Browser erlaubt. Mit Grundkenntnissen in HTML und ein paar zusätzlichen Anweisungen, die in diesem Abschnitt noch vorgestellt werden, sind Sie in der Lage, die Ansteuerungsdateien zu erzeugen. Voraussetzung für die Implementierung der Schnittstelle ist eine ODBC-Verbindung zur Datenbank (die bereits in Kapitel 7 beschrieben wurde) und ein installierter Internet Information Server. Die weiteren notwendigen Arbeitsschritte sind:

1. Starten des Internet Information Servers über START/PROGRAMME/MICROSOFT INTERNET SERVER/INTERNET SERVICE MANAGER. Starten Sie den Dienst in der Menüleiste vom Internet Service Manager (siehe Abbildung 11.31).

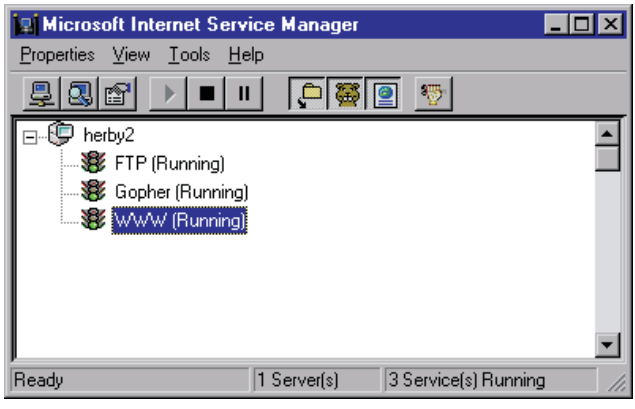


Abbildung 11.31: Starten der Dienste im IIS

2. Geben Sie das Verzeichnis für die Steuerdateien im Dialogfeld für die WWW DIENST EIGENSCHAFTEN frei. Ein Beispiel finden Sie in Abbildung 11.32.

3. Erstellen Sie die HTML-Datei SERVICEVIEW.HTML:

```
<HTML>
<HEAD>
     <TITLE>Sample2Web</TITLE>
</HEAD>
<BODY BGCOLOR="#ffffff">
<H1>Die Web-Datenbank mit Internet Database Connector</H1>
<H2>Geben Sie hier eine Kategorie an und informieren Sie sich über
unser
Service-Angebot:  </H2>
```

```
<FORM METHOD="post" ACTION="http://herby2/sample2web/serv.idc">
Suchbegriff:
<INPUT TYPE="text" NAME="Name" VALUE="" SIZE=30>
<INPUT TYPE="submit" VALUE="Service">
</FORM>
</BODY>
</HTML>
```

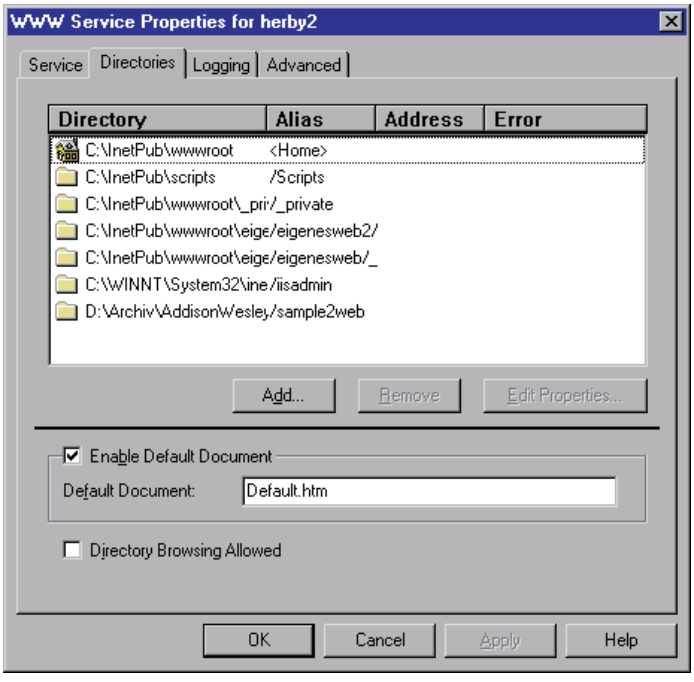


Abbildung 11.32: Freigabe des Verzeichnisses für die Steuerdateien

Innerhalb der FORM-Tags passiert das eigentlich »Aufregende«. Der Benutzer hat die Möglichkeit, einen Suchbegriff einzugeben. Betätigt er die Befehlsschaltfläche SERVICE, dann wird über HTTP der Internet Information Server (IIS) aufgerufen. IIS wertet die Datei SERV.IDC aus. Diese gilt es als Nächstes zu erstellen. Benutzen Sie dafür einfach einen Texteditor.

4. Schreiben Sie die Anweisungen für die IDC-Datei (SERV.IDC) im Texteditor:

```
Datasource: sample sql
Template: serv.htx
SQLStatement:
+ SELECT name AS bezeichnung, beschreibung
+ FROM kategorie
```

```
+ WHERE name LIKE '%%%Name%%%'
+ ORDER BY name
```

Die drei Angaben, die Sie im Beispiel sehen, sind Mindestangaben der IDC-Datei. Als Datenquelle (Datasource) wird der Name für das DSN-System vergeben, den Sie bei der ODBC-Verbindung gewählt haben. Haben Sie ein Kennwort für die Datenbank vergeben, muss dieses ebenfalls angegeben werden. Die SQL-Anweisung wird für die Datenbankabfrage benutzt. Der Parameter `%Name%` wird vom Benutzer im Suchfeld mit einem Wert gefüllt und in der WHERE-Klausel verwendet. Durch die zusätzliche Verwendung von `%%` können auch Teile einer Zeichenfolge oder nur der Anfangsbuchstabe ausgewertet werden. Als Schablone, in welche die Ergebnismenge aus der SQL-Anweisung ausgegeben wird, verwendet man die HTX-Datei. Diese Datei wird im nächsten Schritt erzeugt. Speichern Sie die IDC-Datei aber zunächst in dem Verzeichnis, welches Sie im Internet Information Manager freigegeben haben.

5. Erzeugen Sie die Datei SERV.HTX in einem HTML-Editor:

```
<HTML>
<BODY BGCOLOR="#FFFFFF">
<HEAD>
     <TITLE>Service Ergebnisliste</TITLE>
</HEAD>
<H2>Unser Service-Angebot für Sie:</H2>
<TABLE>
     <TR>
     <TH>Bezeichnung</TH>
     <TH>Beschreibung</TH>
     </TR>
     <%begindetail%>
     <TR ALIGN=RIGHT>
     <TD><%bezeichnung%></TD>
     <TD><%beschreibung%></TD>
     </TR>
     <%enddetail%>
</TABLE>
</BODY>
</HTML>
```

Innerhalb der Tags `<%begindetail%>` und `<%enddetail%>` wird die Ergebnismenge der SQL-Anweisung aus der IDC-Datei ausgewertet. Die Ergebnisspalten werden mit den Tags `<%bezeichnung%>` und `<%beschreibung%>` in das HTML-Dokument eingebettet.

Die volle Funktion können Sie testen, indem Sie in einem Browser das HTML-Dokument SERVICEVIEW.HTML aufrufen. Geben Sie, wie in der Grafik 11.33, einen Suchbegriff an und klicken Sie auf SERVICE. Wenn der Internet Information Server gestartet ist, die ODBC-Verbindung korrekt erstellt wurde, Sie die entsprechenden Rechte für das Verzeichnis haben, in welchem die Steuerdateien enthalten sind und diese die notwen-

digen Einträge enthalten, dann wertet der Internet Information Server die Dateien SERV.IDC und SERV.HTX aus. Anschließend wird das Ergebnis entsprechend der Abbildung 11.34 ausgegeben.

Abbildung 11.33: Suchfunktion über Web-Browser

Die drei notwendigen Felder der IDC-Datei und die optionalen Ergänzungsfelder sind nochmals in den Tabellen 11.2 und 11.3 beschrieben.

Abbildung 11.34: Ergebnis im Web-Browser

Feld	Beschreibung
Datasource	Der Name der ODBC-System-Datenquelle (DSN), der mit dem Programm des ODBC-Administrators erstellt wird.
Template	Der Name der HTX-Datei, welche die Ergebnisdaten der Abfrage formatiert.
SQLStatment	Die SQL-Anweisung, die an die Datenquelle weitergegeben wird. Es kann Parameterwerte des Clients enthalten. Die Parameter müssen zwischen Prozentzeichen (%) stehen. Die SQL-Anweisung kann mehrere Zeilen der IDC-Datei umfassen, von denen jede mit einem Pluszeichen (+) anfangen muss. In einer Datei können auch mehrere SQL-Anweisungen stehen.

Tabelle 11.2: Notwendige Felder in der IDC-Datei

Feld	Beschreibung
Content Type	bestimmt einen gültigen MIME-Typ für die Anzeige der Ergebnisliste der Abfrage.
Default-Parameters	Die optionale Parameterwerte, die in der IDC- Datei verwendet werden, falls ein Parameter vom Client nicht bestimmt wird.

Tabelle 11.3: Optionale Felder in der IDC-Datei

Feld	Beschreibung
Expires	Die Wartezeit in Sekunden, bevor eine Ausgabeseite aus dem Cache aktualisiert wird. Der IDC verwendet den Cache nur, wenn das Feld EXPIRES genutzt wird. Wenn eine folgende Abfrage identisch ist, wird die Seite aus dem Cache geliefert, ohne die Datenbank anzusprechen. Mit diesem Feld lässt sich die Wiederholung einer Datenbankabfrage nach einem bestimmten Zeitintervall einstellen.
MaxField Size	Der maximale Speicherplatz, der einem Feld vom IDC zugewiesen wird. Der Standardwert beträgt 8192 Bytes. Jedes Zeichen, das hinter dieser Begrenzung liegt, wird abgeschnitten.
MaxRecords	Die maximale Anzahl an Datensätzen, die der IDC von einer Abfrage liefern kann. Dieser Wert lässt sich verändern, um die Anzahl zu begrenzen.
ODBCConnection	Durch die Spezifizierung des Wertes POOL für dieses Feld wird die Connection dem Verbindungspool zugefügt, der die Verbindung zur Datenbank für weitere Abfragen aufrechterhält. Dadurch wird die Performance des IDC verbessert. Der IDC schickt nun Daten über eine Verbindung aus dem Pool zur Ausführung einer IDC- Datei, die dieselben Werte für DATASOURCE, USERNAME und PASSWORT enthält. Der Wert NOPOOL bestimmt, dass die Verbindung zu einer .IDC-Datei, in der diese Option steht, nicht aus dem Verbindungspool genommen werden soll.
Password	bestimmt das mit dem Username kombinierte Passwort für den Zugang zu einer ODBC- Datenbank.
Required- Parameters	beinhaltet eine optionale, durch Kommas getrennte Liste von Parameternamen, die HTTPODBC.DLL vom Client benötigt, um sie weiterzuleiten.
TranslationFile	bestimmt einen Pfad zu der Datei, die nichtenglische Zeichen erkennt, so dass sie von Browsern im HTML- Format angezeigt werden können.
UserName	bestimmt einen gültigen Benutzernamen, der für den Zugang zur ODBC-Datenbank benötigt wird.

Tabelle 11.3: Optionale Felder in der IDC-Datei (Forts.)

Mit der SQL-Anweisung, die in der IDC-Datei definiert ist, können auch Datensätze eingefügt, geändert oder gelöscht werden. Verwenden Sie die im Kapitel 4 beschriebenen SQL-Anweisungen, um gewünschte Aktionen zu beschreiben. Mit den Aggregatfunktionen können Sie Ihre Ergebnismengen verdichten, Summen ausgeben oder statistische Berechnungen anstellen.

In der HTX-Datei können Sie neben den HTML-Anweisungen (eine Kurzreferenz finden Sie im Anhang D), im Detailabschnitt die Ergebnisspalten auswerten:

```
<%begindetail%>
    <%ergebnis1%>: <%ergebnis2%>
<%enddetail%>
```

Wird keine Ergebnismenge zurückgegeben, so wird der Detailabschnitt übersprungen. Außerdem kann die Logik mit weiteren Konstrukten erweitert werden. Verwenden Sie Bedingungslogik, um Ihre Ausgabe von bestimmten Werten abhängig zu machen. Das Konstrukt sieht generell so aus:

```
<%if condition%>
     HTML-Anweisungen
[<%else%>
     HTML-Anweisungen]
<%endif%>
```

In der Bedingungsklausel können Sie die in der Tabelle 11.4 aufgelisteten Operatoren verwenden.

Operator	Beschreibung
EQ	wenn der Wert A gleich Wert B ist.
LT	wenn der Wert A kleiner als der Wert B ist.
GT	wenn der Wert A größer als der Wert B ist.
CONTAINS	wenn ein Teil von Wert A den String von Wert B enthält.

Tabelle 11.4: Operatoren für die Bedingungsklausel

In der HTX-Datei können Sie auch direkt Parameter aus der IDC-Datei auswerten. Eine mögliche Anwendung für eine Bedingungslogik mit einer direkten Abfrage des Parameters aus der IDC-Datei können Sie dem Beispiel 11.3 entnehmen.

Beispiel 11.3:

```
<%if idc.name EQ ""%>
<H2>Keine Angabe ist etwas wenig!</H2>
<%endif%>
```

In diesem Beispiel wird der Parameterwert aus der IDC-Datei abgefragt. Wurde kein Wert als Suchbegriff angegeben, so wird ein entsprechender Hinweis ausgegeben. In die HTX-Datei können Sie auch JavaScript oder VBScript implementieren. Damit haben Sie die Möglichkeit, weitere Funktionalität Ihren Web-Seiten zuzufügen.

Beispiel 11.4:

```
<H4>Das aktuelle Abfragedatum:</H4>
<SCRIPT LANGUAGE="VBS">
aktDate = Now
Document.Write aktDate
</SCRIPT>
```

Die Script Engine des IIS verwendet VB Script auf dem Server und unterstützt das Senden von VBScript und Javascript an den Browser. Die Skriptsprachen bringen viele zusätzliche Gestaltungsmöglichkeiten, zudem ist der Umgang mit diesen Sprachen nicht sonderlich schwer zu erlernen.

Abschließend sollen die wichtigen Schritte für die Web-Anbindung über den Internet Database Connector benannt werden:

- HTML-Datei:
  - Schreiben Sie den HTML-Code für die Übergabe der URL und der Parameter an den Server.
- IDC-Datei:
  - Spezifizieren Sie Datenquelle, Sql-Anweisungen und die HTX-Datei.
- HTX-Datei:
  - Ergebnismenge für den Browser aufbereiten.

12 Multimediale Daten

Multimedia ist auf dem Vormarsch. Für diese Erkenntnis muss man kein Insider und kein Prophet sein, ein Navigieren durch das Internet führt einem diese Tatsache eindrucksvoll vor Augen. Für den sinnvollen Einsatz von Multimedia genügt es nicht, einen Rechner sprechen oder Videobilder erzeugen zu lassen. Die Medien Ton und Videobild müssen vollständig und gleichberechtigt in die Rechnertechnologie integriert werden. Daher wird statt Multimedia auch der Begriff Medienintegration verwendet, weil dieser das konkrete Entwicklungsziel besser beschreibt. Ein Computer soll zum Beispiel zur Erstellung, Speicherung, Bearbeitung und Ausgabe von bewegten Grafiken, Realbild- und Tonfolgen dieselben Möglichkeiten bieten, die er für die Bearbeitung von Texten, Grafiken und Einzelbildern zur Verfügung stellt. Was für den Bereich der Datenbankprogrammierung heißen würde, dass es z.B. für Videosequenzen ebenso möglich sein sollte, sie in der Datenbank zu organisieren und abzurufen, wie dies auch bei einfachen Textinformationen der Fall ist. Selbstverständlich ist die Art und die Menge der Daten ungleich höher, wenn es darum geht, multimediale Inhalte zu verwalten. Herkömmliche Strategien und Techniken bleiben hier auf der Strecke und so wurden neue Konzepte entwickelt.

12.1 Was sind multimediale Daten? – Definition

Wann kann man eigentlich von multimedialen Daten sprechen? Ist es schon Multimedia, wenn einige Textstellen ein wenig blinken oder auf einer Internetseite ansprechende Grafiken flimmern? Von vielerlei Seiten wurde auf diese Frage Antwort gegeben. Berufene und Unberufene haben mehr oder weniger schwammige Definitionen geliefert. Tatsächlich ist es nahezu unmöglich, eine messerscharfe Definition für diesen Begriff zu liefern durch die Eigendynamik, die diese Technik ausgelöst hat, wurden in vielen Bereichen Weiterentwicklungen vorangetrieben und mit diesem Begriff geworben. Auf einmal war alles Multimedia, das Telefonbuch auf CD, die vielen Journals im Internet oder das Navigationssystem im Auto, das mit beruhigender Stimme den gestressten Autofahrer durch den »Vorstadtdschungel« führen soll. Einige Punkte sollen umschreiben, wie Multimedia wirkt und was benötigt wird:

- Es werden mehrere unserer Sinne gleichzeitig angesprochen.
- Es handelt sich oft um kontinuierliche Medien wie Sound oder Video.
- Es werden hohe Anforderungen an die Technik (Rechnerleistung, Bandbreite, Speicher) gestellt.
- Es erfordert oft die Synchronisation mehrerer Datenströme.

Bei Multimedia wirken also mehrere Medien zielgerecht zusammen. Dies erfordert hohe Ansprüche an die verwendete Technik. Es werden nicht nur sehr viel mehr Daten verarbeitet, sondern auch Daten unterschiedlicher Beschaffung. Durch die Verschiedenartigkeit der Daten treten viele Probleme auf, da aufgrund ihrer Charakteristik unterschiedliche Speicherungstechniken und daher auch unterschiedliche Zugangsstrukturen existieren. Daher müssen für jede Bearbeitungsmethode einige grundlegende Auswahlen getroffen werden:

- verwendete Zugiffs- und Speicherungsmethode,
- die Art der Anfrageprädikate,
- die Wahl von aktiven und passiven Komponenten,
- Annäherung für die Definition der Datenstrukturen und der Berechnungen.

12.1.1 Hypertext und Hypermedia

Schon 1969 stellten in den USA *Nelson* und *Van Dam* einen Hypertext-Editor vor, 1976 wurde der Begriff »Multiple Media« geprägt.

Hypertext ist ein Text, der Verweise zu anderen Texten enthält – wie es auch bei einem Lexikon der Fall ist. Hypertext ist daher nicht-linear, denn man kann zwischen den Texten »herumspringen«. Insbesondere für kontextabhängige Hilfesysteme und Online-Handbücher wird Hypertext schon lange eingesetzt. Das DOS- oder Windows-Hilfesystem ist ein Hypertextsystem.

Hypermedia ist nicht auf Text beschränkt. Hypermedia-Dokumente können andere Medien enthalten, z. B. Diagramme und Bilder. Dabei gibt es keine Beschränkung auf statische Medien, sondern auch Sounds, Musik, Animationen und Videoseqenzen lassen sich einbinden. Das World Wide Web ist das beste Beispiel für Hypermedia.

12.1.2 Grafikformate

Die verschiedenen weitverbreiteten Grafikformate mit ihren Besonderheiten seien im Folgenden einmal aufgeführt.

BMP

Das Format BMP wird von den meisten Grafikprogrammen unterstützt, die unter MS-Windows arbeiten. Auch die meisten Konvertierprogramme unterstützen BMP.

Bezeichnung	**Microsoft Windows Bitmap, BMP, DIB**
Farben	1-Bit (s/w), 4-Bit (16 Farben), 8-Bit (256 Farben), 24-Bit (16,7 Mio. Farben)
Kompression	normalerweise keine oder RLE
Maximale Bildgröße	65536 x 65536 Pixel

Tabelle 12.1: Das Grafikformat Bitmap

GIF

GIF wurde von den Firmen UNISYS Corp. und CompuServe entwickelt. Ziel war eine minimale Dateigröße zum Austausch von Graphiken über Mailboxen. Da im Mikrocomputerbereich für jede gängige Hardware (Amiga, Atari, IBM-kompatible, Macintosh) Programme existieren, die GIF-Grafiken verarbeiten können, ist es vor allem als Austauschformat über Hardwaregrenzen hinweg von Bedeutung. Eine GIF-Datei kann mehrere Bilder enthalten, was z. B. Interlacing ermöglicht. Im sequentiellen Modus wird das Bild zeilenweise von links oben nach rechts unten codiert und ausgegeben. Unter »Interlacing« versteht man eine Abwandlung des Bildaufbaus bei der Wiedergabe.

Bezeichnung	**Graphics Interchange Format, GIF**
Farben	1- bis 8-Bit (s/w bis 256 Farben/Graustufen)
Kompression	LZW
Maximale Bildgröße	65536 x 65536 Pixel

Tabelle 12.2: Das Grafikformat GIF

JPEG

JPEG ist die Abkürzung von Joint Photographic Experts Group, wird von ISO standardisiert (ISO/IEC JTC1/SC2/WG10) und beschreibt Algorithmen zur Kompression digitaler Bilder. Die Kompression nützt Redundanz und Eigenschaften des Auges bei der Interpretation der Bilder aus. Der Kompressionsgrad ist einstellbar, da mit zuneh-

mender Kompression Information verloren geht. Kompressionsfaktoren von über 100 sind erreichbar und werden zur Erzeugung von Symbolen (Icons) aus Rasterbildern benutzt.

Bezeichnung	**JPEG File Interchange Format, JPG, JPEG, JFIF, JFI**
Farben	Bis 24-Bit (bis 16,7 Mio. Farben)
Kompression	JPEG
Maximale Bildgröße	65536 x 65536 Pixel

Tabelle 12.3: Das Grafikformat JPEG

PCX

Das PCX-Format wurde von der Firma ZSoft Corporation zur Speicherung und Übertragung der mit PC-Paintbrush erstellten Grafiken entwickelt. Dieses Format wurde von Microsoft übernommen und u.a. im Grafikprogramm MS-Paintbrush für Windows benutzt. PCX ist im PC-DOS/Windows-Bereich weit verbreitet, aber auch auf anderen Plattformen anzutreffen.

Nachteile des PCX-Formates sind eine hardwareabhängige Darstellung von Farben und Auflösung sowie ein relativ ineffizienter RLE-Kompressionsalgorithmus.

Bezeichnung	**PC Paintbrush File Format, DCX, PCC**
Farben	1-, 2-, 4-, 8-, 24-Bit (s/w bis 16,7 Mio. Farben)
Kompression	Keine oder RLE
Maximale Bildgröße	65536 x 65536 Pixel

Tabelle 12.4: Das Grafikformat PCX

TIFF

TIFF ist eine Entwicklung der Firma Aldus Corporation. Dieses Format hat sich in den letzten Jahren zu einem der wichtigsten Formate für Rasterdateien entwickelt. Es wurde von Anfang an so umfangreich konzipiert, dass es eine Vielzahl von Speichermöglichkeiten bietet und neben den eigentlichen Grafikdaten auch Angaben wie der Name der benutzten Grafiksoftware oder der Scannertyp aufgenommen werden können. TIFF ist in der Lage, Schwarz/Weiß-, Grauwert- und Farbbilder zu speichern. Diese Möglichkeiten machen das Format komplizierter, andererseits aber auch universeller einsetzbar. Neben den meisten Scannern benutzen viele Grafikprogramme das TIFF-Format. Das Einlesen von TIFF-Bildern bereitet manchen Programmen große Probleme. Aufgrund der möglichen Komplexität einer TIFF-Datei und der damit verbundenen Varianten (z.B. viele verschiedene Kompressionsmethoden) lesen viele Pro-

gramme nur einen kleinen Anteil aller TIFF-Varianten. Die Fehlerursache ist dabei aber meist bei diesen Programmen zu suchen, da das TIFF-Format sehr präzise definiert ist.

Bezeichnung	**Tag Image File Format, TIF**
Farben	1- bis 24-Bit (s/w bis 16,7 Mio. Farben)
Kompression	Keine, RLE, LZW, CCITT Group 3 und 4, JPEG
Maximale Bildgröße	ca. 4 Milliarden Bildzeilen

Tabelle 12.5: Das Grafikformat TIFF

12.1.3 Audioformate

Es gibt mittlerweile viele Dateiformate, um digitalisierte Audiodaten zu speichern. Sie unterscheiden sich insbesondere im verwendeten Kompressionsalgorithmus und in der Bandbreite. Drei »prominente« Vertreter sollen im weiteren Verlauf vorgestellt werden.

AU-Format

Dieses Format wurde von SUN Microsystems entwickelt. Es ist sehr einfach aufgebaut und wird von nahezu allen UNIX-Workstations und allen World-Wide-Web-Browsern unterstützt. Normalerweise werden die Daten in 8 bit codiert. Die Original-SUNs unterstützen nur eine Abtastrate von 8 KHz, weshalb diese Rate häufig verwendet wird. Dateien im AU-Format bestehen entweder aus Header und Datenteil oder nur aus den rohen Audiodaten.

WAV

Das WAV-Format von Microsoft basiert auf dem EA-IFF-85-Standard (IFF = Interchange Format Files), der von Electronic Arts entwickelt wurde. Ein wesentlicher Unterschied besteht jedoch in der Anordnung der Bytes (INTEL-Anordnung: LSB, MSB). WAV ist ein Teil des allgemeinen RIFF-Standards (Resource Interchange File Format) von Microsoft. So existieren neben WAV auch Dateien für die Speicherung von Bitmaps, Farbpaletten und zwei Formate für MIDI-Daten. Neue Chunk Typen können nur von Microsoft festgelegt werden. Unbekannte Chunk-Typen müssen ignoriert werden. Durch die Integration in Microsoft-Windows dürfte WAV das am weitesten verbreitete Format für Audiodaten sein.

MPEG-1-Audio

MPEG-1-Audio ist ein Standard des MPEG-Komitees, das vor allem für seinen Standard für die Digitalisierung von Bewegtbildern bekannt ist (MPEG = Moving Pictures Experts Group). MPEG-1-Audio erreicht Kompressionsraten bis zu 1:22 ohne größere

hörbare Verluste. Üblich sind Werte von 1:6 oder 1:7. Aufgrund dieser hohen Kompressionsraten wird es unter anderem für den kommenden Standard für digitale Rundfunksendungen in Europa verwendet (Eureka-147).

Das Format von MPEG-1 Audio ist zu komplex, um es hier komplett vorzustellen, daher wird im Folgenden vor allem auf die Prinzipien eingegangen.

MPEG-1 Audio unterstützt maximal 2 Audiokanäle. Die Abtastrate kann 44.1, 48 oder 32 kHz betragen. Das Format ist bitorientiert, eine Ausrichtung auf Bytes findet im Allgemeinen nicht statt. Die Audiodaten werden hintereinander in Frames übertragen, die jeweils abhängig vom Level eine feste Anzahl Abtastpunkte aufnehmen. Bei Level II besteht beispielsweise ein Frame aus 1152 Abtastpunkten, was bei 44.1 kHz einem Ausschnitt von 26 ms entspricht. Die Frames besitzen jeweils neben den Audiodaten zusätzliche allgemeine Informationen und können mit einer 16-Bit-Prüfsumme gesichert werden.

12.1.4 Videoformate

Videodaten benötigen hohe Speicherungskapazitäten und besondere Algorithmen, um verschiedene Datenströme zu synchronisieren. Daten wie Audio und Video erfordern zudem sehr hohe Datenraten zur Abspielzeit.

AVI-Format von Microsoft

Das AVI (Audio Video Interleaved) Dateiformat ist eine spezielle Ausführung der von Microsoft für Windows 3.1 eingeführten Multimediaerweiterung: dem RIFF (Resource Interchange File Format). Es dient dazu, auf rekonstruierbare Art und Weise Audio-Videoinformationen zu speichern. Eine AVI-Datei besteht wie schon etliche Grafikformate) aus Chunks, also aus verschiedenen, ineinander verschachtelten Datenstrukturen. Sie können, jeder für sich, in weitere Strukturen (Sub-Chunks) unterteilt werden.

Die einfachste Form einer AVI Datei wäre ein einziger Videostream, meistens wird man aber dazu noch Audio-Informationen speichern wollen. Allgemein ist es möglich, auch spezielle Multimedia-Sequenzen wie etwa einen MIDI-Track als zusätzlichen Datenstream in einer AVI-Datei aufzunehmen. Um Audioinformationen in Stereoton zu sichern, benötigt man dazu nur einen Stream, genauso als würde nur Mono gespeichert werden. Weiterhin sind Control-Tracks möglich, etwa um externe Geräte zu steuern.

MPEG

Die Motion Picture Expert Group (MPEG) wurde Ende der 80er Jahre zur Festlegung eines digitalen Standards für Bewegtbilddarstellung ins Leben gerufen. Die MPEG-Verfahren definieren Normen für Computer-gestützte Bewegtbilddarstellung, welche

für die Bereiche Heimanwendung (MPEG-1) und Studio (MPEG-2) entwickelt wurden. Zentraler Bestandteil des MPEG-Codierungsalogorithmus ist die so genannte Bewegungskompensation. MPEG nutzt hierbei die Tatsache, dass sich zwei aufeinanderfolgende Bilder derselben Filmszene in der Regel kaum unterscheiden. Anstatt also zwei Vollbilder mit hohem Speicheraufwand zu codieren, werden lediglich die veränderten Bilddetails erfasst und weiterverarbeitet. Verwendung finden hierbei drei verschiedene Bildtypen (Intra-, Predicted- und Bidirectional predicted), welche sich in Funktion und Codierung voneinander unterscheiden. Bis zur Verabschiedung der Norm MPEG-1 standen bereits verschiedene Verfahren zur Verfügung (Motion-JPEG (M-JPEG) und die Recommendation H.261 der CCITT).

Das M-JPEG-Verfahren basiert auf der Serialisierung von im JPEG-Format komprimierten Einzelbildern. Die Qualität der verlustbehafteten JPEG-Kompression ist dabei in weiten Grenzen konfigurierbar. Der Kompressionsgrad liegt in der Einstellung für gute Bildqualität bei einem Faktor von 20–25. Die Einbindung von Audiodaten ist jedoch nicht im M-JPEG definiert. Eine Norm für M-JPEG existiert nicht. Durch die Fixierung auf Einzelbildkompression erwies sich das Verfahren als zu ineffizient.

Mit der Recommendation H.261 stand ein audiovisueller Standard der CCITT zur Verfügung. H.261 besitzt durch seine Ausrichtung auf Bewegtbildübertragung bereits einen hybriden Kompressionsalgorithmus, der aus einer Spielart des Deltaverfahrens, einer Restbildkompression sowie einem optional einbindbaren Verfahren zur Bewegungskompensation besteht. Der Standard ist u. a. in Auflösung und Bildrate (7,5–30 Bilder/s) konfigurierbar, wobei sich die Bandbreite im Bereich von 40 KBit/s bis zu 2 MBit/s bewegt. Die relativ enge Auslegung der Recommendation erwies sich jedoch als starkes Handicap.

Eine Lösung für die oben aufgeführten Probleme steht mit MPEG-1 und MPEG-2 zur Verfügung, wobei wesentliche Merkmale sowohl von M-JPEG als auch H.261 in die Normung eingingen. MPEG wurde dabei als generische Norm entworfen, wodurch eine Anpassung an verschiedene Anwendungen möglich gemacht werden soll.

Die ältere MPEG-1 Norm findet Anwendung in der PC-Welt und im Unterhaltungsmarkt. Die Übertragungsrate liegt bei den dort üblichen Auflösungen im Bereich bis zirka 1,5 MBit/s (192 KByte/s), wobei aus praktischen Gründen die Grenzen des MPC-1-Standards (für single-speed CD-ROM Laufwerke) eingehalten werden. Neben der CD-ROM eigenen sich auch andere Medien als Basis für MPEG. Hierzu gehört das DAT-System als auch die Verwendung in LANs zur Unterstützung von Video-Mail oder Video-Übertragungen über Telefonleitungen (ISDN). MPEG 2 wurde im Hinblick auf digitales Fernsehen definiert, es unterstützt das Zeilensprungverfahren (Interlace). Die der Kompression zugrundeliegenden Verfahren wie Bewegungskompensation, DCT, Quantisierung und Huffman-Codierung sind für beide MPEG-Normen im Prinzip identisch. Verbesserungen erfolgten lediglich auf Detailebene. MPEG 2 stellt also

von den Grundprinzipien her keinen größeren Fortschritt gegenüber MPEG 1 dar, insbesondere löst MPEG 2 nicht MPEG 1 ab. So erreichen beide Verfahren in der Regel identische Datenverdichtungen.

MPEG beschreibt, wie komprimierte Video- und Audio-Daten zu einem Bitstrom zusammengefasst werden. Die Norm umfasst zwei Teile: die Systemschicht und die Kompressionsschicht. Diese Schichten haben ihrerseits wieder Teilschichten.

Die Funktion der Systemschicht definiert, auf welche Weise Bitströme von Video- und Audio-Daten sowie optional privater Daten zu einem einzigen Bitstrom zusammengefasst werden. Außerdem werden Zeitmarken (Presentation Time Stamp) bereitgestellt, die der synchronisierten Wiedergabe von Bild und Ton dienen. Der maximale Abstand zwischen zwei Synchronisationspunkten liegt bei 0,7 Sekunden. Die Systemschicht ist unterteilt in den Pack-Layer und den Packet-Layer. Innerhalb eines Packs werden Packets aneinandergereiht. Packets enthalten neben den Headerinformationen nur Daten einer einzigen Art, also entweder Audio-Daten, Video-Daten oder private Daten.

Diese Daten werden von den Kompressionsschichten für Video und Audio codiert und zur Übertragung in die Päckchen eingefügt.

Das MPEG-Verfahren nutzt die Tatsache, daß in Folgen bewegter Bilder zwischen aufeinanderfolgenden Bildern große Ähnlichkeit besteht. Mit der Ausnahme krasser Szenenwechsel werden sich Bilddetails kontinuierlich von einem Bild zum nächsten fortsetzen, wie zum Beispiel eine sich von links nach rechts bewegende Person. Ein zentraler Bestandteil von MPEG ist nun die so genannte Motion Compensation: Die Bewegung der Person wird einfach durch einen Vektor beschrieben, zum Beispiel durch die Angabe, daß sie sich von einem Bild zum nächsten um 12 Pixel nach rechts und um 10 Pixel nach oben bewegt hat.

Die Erkennung eines zusammengehörigen Objekts wäre in der Praxis allerdings viel zu aufwändig. Stattdessen werden so genannte Makroblöcke mit einer Pixelgröße von 16 x 16 untersucht. Diese Makroblöcke entsprechen vier Blöcken, wie sie bei JPEG codiert werden. Im nächsten Schritt wird die Differenz aus dem realen Makroblock in Filmbild 1 und dem verschobenen Makroblock aus Filmbild 2 gebildet, Dieses Fehlerbild muss neben dem Verschiebungsvektor zur Beobachtung der Fehlerfortpflanzung codiert und gespeichert werden. Der geringste Speicheraufwand entsteht natürlich, wenn der Unterschied zwischen den verschobenen Makroblöcken und den tatsächlich dargestellten Blöcken so klein ist, daß auf die Codierung der Differenz ganz verzichtet werden kann.

MPEG steuert die Darstellung von komprimiertem Video durch die Festlegung einer Syntax. Die Regeln zur Erfassung der Bewegungskompensation lassen hingegen viele Freiheiten zu, so daß die Qualität des MPEG-Endprodukts auch maßgeblich von der Güte des verwendeten Codierungs-Algorithmus abhängt.

12.2 Datentypen für multimediale Daten

Das vorige Kapitel hat gezeigt, dass wir bei multimedialen Daten viel komplexere Strukturen haben als dies bei »herkömmlichen« Daten der Fall gewesen ist. Bei Multimedia- oder auch CAD-Daten stoßen wir mit den bisher kennen gelernten Datentypen auf Grenzen. In der Norm SQL-99 finden wir deshalb auch weitere Datentypen, die für Objekte mit besonderes hohem Strukturaufwand geeignet sind. Es handelt sich dabei um die so genannten *LOB-Typen*.

LOB steht für Large Objects. Grundsätzlich kann man zwei LOB-Typen unterscheiden. Geht es um eine große Menge lesbarer Zeichen, so verwendet man dafür *CLOBs* (Character Large Objects). Bei multimedialen Daten haben wir es jedoch eher mit Binärdaten zu tun. Grafiken, Videos oder auch Programmcodes werden deshalb in *BLOBs* (Binary Large Object) abgelegt. Ein *BLOB* kann durchaus mehrere Gigabyte an Daten umfassen.

In Abhängigkeit von der jeweiligen Plattenbelegung kann es, besonders bei größeren BLOBs, zu starker Fragmentierung eines einzelnen Datensatzes und damit zu längeren Zugriffszeiten kommen. Bei Datenbanken mit Transaktionsprotokollierung werden BLOB-Daten mit in das Transaktionsprotokoll geschrieben. Dies führt zu hohen Belastungen des Gesamtsystems während einer Transaktion.

Werden die BLOB-Daten in einem separaten Speicherbereich abgelegt, so lassen sich diese Probleme reduzieren, da sich die zugewiesenen Speichereinheiten an den tatsächlichen Bedarf ausgerichtet werden können und im Transaktionsprotokoll nur die Verweise auf die BLOBs erscheinen.

Eine weitere Kategorie bilden die *Bfiles*; dies sind Dateien außerhalb des Datenbanksystems, in denen Daten abgelegt und bei Bedarf mit eingebunden werden.

Die beschriebenen Datentypen waren schon vor dem Standard SQL-99 in vielen DBMS einsetzbar. Daten vom Typ LOB finden wir deshalb schon in früheren Versionen von Oracle, DB2 oder dem SQL Server.

Beim SQL Server kann man auf die Standarddatentypen *ntext*, *text* und *image* zurückgreifen, um große Objekte abzulegen. In Tabelle 12.6 finden Sie eine Kurzbeschreibung zu diesen drei Datentypen.

Datentyp	**Art der Daten**	**Maximale Länge**
ntext	Unicode-Daten	2^30 – 1 (1.073.741.823) Zeichen
text	Nicht-Unicode-Daten	2^31 – 1 (2.147.483.647) Zeichen
image	Binärdaten	2^31 – 1 (2.147.483.647) Bytes

Tabelle 12.6: LOB-Datentypen beim MS SQL Server

Beim SQL Server können LOB-Daten standardmäßig außerhalb der Tabelle auf separaten Datenseiten, mit einer Zeigerstruktur auf diese Seiten, gespeichert werden oder optional ebenso wie die anderen Daten der gleichen Zeile auf derselben Datenseite.

PostgreSQL bietet eine Palette geometrischer Datentypen, die sich sehr gut für CAD-Daten eignen. Die Datentypen beschreiben jedoch ausschließlich zweidimensionale Objekte. Die Tabelle 12.7 zeigt die geometrischen Datentypen von PostgreSQL.

Datentyp	**Darstellung**	**Maximale Länge**
point	(x,y)	16 Bytes
line	((x1,y1),(x2,y2))	32 Bytes
lseg	((x1,y1),(x2,y2))	32 Bytes
box	((x1,y1),(x2,y2))	32 Bytes
path	((x1,y1),...)	4+32n Bytes
polygon	((x1,y1),...)	4+32n Bytes
circle	<(x,y),r>	24 Bytes

Tabelle 12.7: Geometrische Datentypen bei PostgreSQL

Auch wenn MS Access keine ausgewiesene Multimediadatenbank ist, so ist es auch hier möglich, Hyperlinks oder Grafiken in der Datenbank zu verwalten. Um Daten, wie z. B. Microsoft Word- oder Microsoft Excel-Dokumente, Bilder, Klänge und binäre Daten zu speichern, können Felder des Datentyps OLE-Objekt (OLE: Object Linking and Embedding) verwendet werden. OLE-Objekte können mit einem Feld in einer Microsoft Access-Tabelle verknüpft oder in ein Feld eingebettet werden. Verwenden Sie in einem Formular oder Bericht ein Steuerelement, um das OLE-Objekt anzuzeigen. Ein verknüpftes OLE-Objekt zeigt dessen Daten in einem Formular oder Bericht an, doch die Daten bleiben in der Originaldatei gespeichert. Sie können ein verknüpftes OLE-Objekt aus dem Formular bzw. Bericht heraus bearbeiten und Änderungen an der Originaldatei speichern, doch dieses kann weiterhin aus dem OLE-Server geöffnet und bearbeitet werden. Wenn die Informationen im Objekt durch den OLE-Server geändert werden, spiegelt sich die Änderung in Microsoft Access wider. Im Gegensatz dazu stellt ein eingebettetes OLE-Objekt eine Kopie der im OLE-Server erstellten und in der Datenbankdatei gespeicherten Informationen dar. Änderungen an der Originaldatei werden in Microsoft Access nicht berücksichtigt, ebenso wie Änderungen am eingebetteten Objekt in der Originaldatei nicht gespeichert werden. Im nächsten Abschnitt werden Sie erfahren, wie Sie Grafiken m.H. des OLE-Objektes in der Beispieldatenbank verwalten können.

Das Hyperlink-Objekt repräsentiert einen *Hyperlink*, der mit dem Steuerelement eines Formulars, Berichts oder einer Datenzugriffsseite verknüpft ist. Wenn Sie eine Tabelle durch Importieren von Daten erstellen, macht Microsoft Access aus jeder Spalte, die URLs oder UNC-Pfade enthält, automatisch ein Hyperlink-Feld. Eine Spalte wird jedoch nur umgewandelt, wenn alle Werte mit einem bekannten Protokoll beginnen, wie z. B. »http:« oder »\\«. Beginnt nur ein einziger Wert mit einem unbekannten Protokoll, wird die Spalte nicht in ein Hyperlink-Feld umgewandelt.

12.3 Große Datenobjekte in der Beispieldatenbank

Ausgangspunkt für die illustrative Beschreibung der Verwaltung von Multimediadaten in der Datenbank soll wieder die Beispieldatenbank sein. Wir verwenden für dieses Beispiel MS Access und Visual Basic. Bei dem VB-Programm handelt es sich um eine einfache Anwendung, die Grafikdateien laden, anzeigen und als OLE-Objekt in der Datenbank ablegen kann. Zusätzliche Funktion ist das »Blättern« in den vorhandenen Bildern der Datenbank. Verwenden Sie Visual Basic 5.0 oder 6.0 und MS Access 97 oder 2000.

12.3.1 Erzeugen der Tabelle mit multimedialem Datentyp

Bisher ist in der Beispieldatenbank noch keine Tabelle für Grafiken vorgesehen. Diese Tabelle soll nun erzeugt werden. Für die verschiedenen Serviceangebote sollen jeweils Beschreibungsbilder in der Datenbank abgelegt werden. Für die neue Tabelle *bilder* werden die Felder *bildid* (Autowert), *titel* (Text), *kategorie*(Text) und *bildblob* (OLE) benötigt. Führen Sie dazu folgende Schritte aus:

1. Öffnen Sie die Beispieldatenbank in MS Access.
2. Legen Sie eine neue Tabelle an EINFÜGEN/TABELLE.
3. Wählen Sie im Dialogfeld NEUE TABELLE die ENTWURFANSICHT.
4. Definieren Sie die notwendigen Felder, wie dies in Abbildung 12.1 zu sehen ist.
5. Beschreiben Sie auch die neu definierten Felder.
6. Schließen Sie die Entwurfsansicht und vergeben Sie den Namen für die Tabelle.

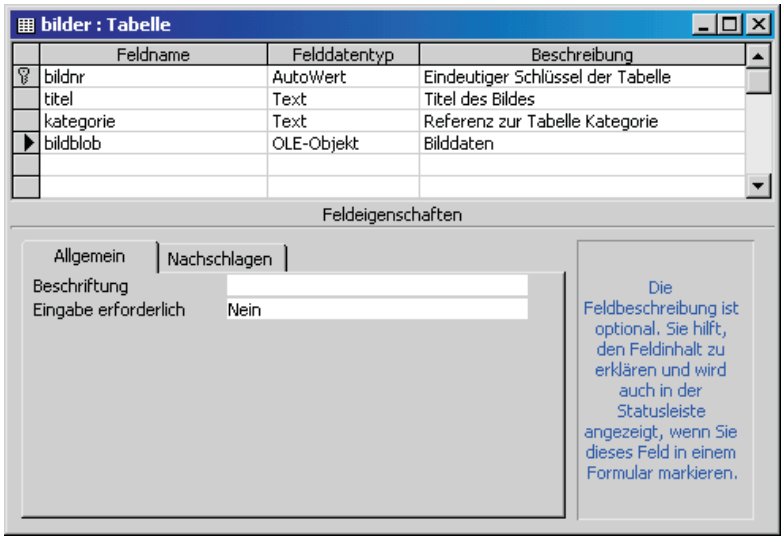


Abbildung 12.1: Felddefinition in MS Access

12.3.2 Bilder in die Tabelle einfügen

Bei BLOBs ist ein einzelner Datenwert normalerweise zu groß, um von einer Anwendung in einem Schritt abgerufen werden zu können. Die Werte können sogar zu groß für den auf dem Client verfügbaren virtuellen Arbeitsspeicher sein. Deshalb sind normalerweise spezielle Schritte notwendig, um diese Werte abzurufen. Zum Lesen einer Spalte mit langen Daten schließt die Anwendung die Spalte, die BLOBs enthält, einfach in eine Auswahlliste ein und bindet die Spalte dann an eine Programmvariable, die groß genug ist, um einen angemessenen Block der Daten zu speichern. Die Anwendung führt dann die Anweisung aus und verwendet eine API-Funktion oder -Methode, um die Daten blockweise in die gebundene Variable abzurufen. *ADO* kann Spalten oder Parameter mit multimedialen Daten zu einem Field- oder Parameter-Objekt zuordnen. Mit den Methoden *GetChunk* und *AppendChunk* können die Daten blockweise abgerufen und geschrieben werden. Zum Einfügen der Bilddaten brauchen wir also zunächst die Funktion *AppendChunk*. *AppendChunk* hängt Daten an ein Field- oder Parameter-Objekt an, das langen Text oder Binärdaten enthält.

Die Syntax der Methode AppendChunk:

```
Objekt.AppendChunk Data
```

Dabei ist das *Objekt* vom Typ *Parameter* oder *Field*. Bei *Data* handelt es sich um einen Wert vom Typ *Byte*, der die Daten enthält, die an das Objekt angehängt werden sollen.

Die Bilddaten sollen also aus einer Grafikdatei (z.B. JPG, GIF oder BMP) ausgelesen werden. Den Beispielcode für diese Aktion sehen Sie im Beispiel 12.1.

Beispiel 12.1:

```
Private rs As ADODB.Recordset
Dim bytBLOB() As Byte
Dim intNum As Integer
Dim strImagePath As String
intNum = FreeFile
Open strImagePath For Binary As #intNum
ReDim bytBLOB(FileLen(strImagePath))
Get #intNum, , bytBLOB
Close #intNum
rs.Fields("bildblob").AppendChunk bytBLOB
rs.Update
```

In den ersten vier Zeilen werden die Variablen dimensioniert. Mit der Funktion *FreeFile* wird die nächste verfügbare Dateinummer der Variable *intNum* übergeben. Die Datei, deren Name zuvor im Programm mit Namen und Verzeichnis gefüllt worden sein muss, würde für den binären Zugriff geöffnet. Anschließend wird mit *ReDim* der Speicherplatz der Variablen *bytBlob* an den tatsächlich benötigten Platz angepasst. Die Anweisung `Get` liest die Daten aus der geöffneten Datei. Die geöffnete Bilddatei wird dann geschlossen. Mit der Methode *AppendChunk* werden die binären Daten dem Recordset übergeben und später der gesamte Datensatz in die Datenbank geschrieben. Den gesamten Programmcode finden Sie auf der Begleit-CD.

12.3.3 Bilder abrufen

Für das Lesen der Bilddaten aus der Datenbank bedienen wir uns auch wieder der Methoden von ADO und der Datenbankzugriffsstrategie OLE DB. Zum Lesen der Bilddaten wird die Funktion *GetChunk* benötigt. Die Funktion gibt den gesamten Inhalt oder eine Teilmenge des Inhalts eines Field-Objekts zurück, das langen Text oder Binärdaten enthält.

Die Syntax der Methode GetChunk:

```
Variable = Feld.GetChunk( Size )
```

Size ist ein Long-Ausdruck, der der abzurufenden Byte- oder Zeichenanzahl entspricht. Den Programmcode für das Abrufen der Bilddaten aus der Datenbank finden wir in dem Beispiel 12.2.

Beispiel 12.2:

```
Private rs As ADODB.Recordset
Dim lngImageSize As Long
Dim lngOffset As Long
Dim bytChunk() As Byte
Dim intFile As Integer
Dim strTempPic As String
Const conChunkSize = 100
intFile = FreeFile
Open strTempPic For Binary As #intFile
lngImageSize = rs("bildblob").ActualSize
Do While lngOffset < lngImageSize
bytChunk() = rs("bildblob").GetChunk(conChunkSize)
Put #intFile, , bytChunk()
lngOffset = lngOffset + conChunkSize
Loop
Close #intFile
```

Der eigentliche Lesevorgang erfolgt in der DO WHILE-Schleife. Die GetChunk-Methode liest die Daten aus der Datenbank und übergibt sie der Variable *bytChunk*. Die Daten werden dann erst in eine temporäre Datei geschrieben, bevor sie in ein geeignetes Steuerelement übertragen werden. Auch diese Programmcodes finden Sie als Visual Basic-Projektdatei auf der Begleit-CD.

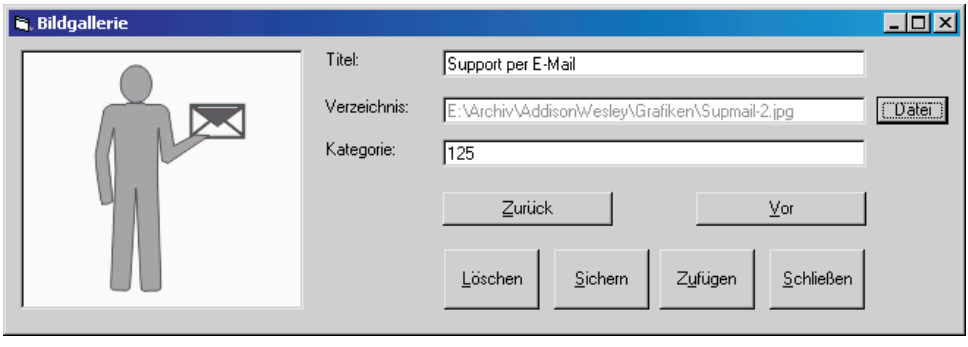


Abbildung 12.2: Bildbetrachter als VB-Programm

Mit dem kleinen VB-Programm sind Sie nun in der Lage, Grafiken in die Datenbank zu schreiben und diese wieder abzurufen. Ohne größeren Aufwand bei der Anpassung, kann das Programm auch für den SQL Server verwendet werden. Auch beim SQL Server sind die Methoden *AppendChunk* und *GetChunk* für Input und Output von BLOBs geeignet.

12.4 Schneller Zugriff auf multimediale Daten

Bei der Anfrage sollte das System in der Lage sein, Objekte direkt und effizient zu lokalisieren. Dies kann erreicht werden durch Zuweisen von zusätzlichen Datenstrukturen für die verschiedenen Datenarten und durch Zugriffsmethoden, die zufälligen Zugriff unterstützen. Für verschiedene Typen können daher verschiedene Strukturen sinnvoll sein. Für jede Art von Daten müssen die Zugriffsmethoden definiert und kombiniert werden. Insbesondere:

Formatierte Daten:

Formatierte Daten werden charakterisiert durch Objekte. Durch einen Satz von Attributen (Suchschlüssel) wird jedes Objekt betrachtet. In Anbetracht verschiedener Suchschlüssel können auch verschiedene Suchmethoden konstruiert werden. Die Zugriffsmethode hängt stark vom Datenmodell der Datenorganisation ab. Für formatierte Daten gibt es zwei Annäherungen: Index und *Hash Funktionen*. Index Funktionen unterstützen den direkten Zugriff wohingegen Hash Funktionen die Berechnung der Adresse eines Objektes erfordern.

Text:

Textobjekte variieren in ihrer Länge, und ihr Inhalt besteht aus Zeichen aus einem sehr großen Zeichenvorrat. Anfragen auch Textobjekte sind selten zufriedenstellend, daher muss man Punkte finden, welche die Anfrage wenigstens zum Teil zufriedenstellen. Zurückholtechniken wurden für exakte und partielle Fragen definiert. Exakte Zurückholmethoden basieren auf der boolschen Logik. Die am häufigsten verwendeten *Zurückholmethoden* sind *Volltextsuche*, *Umkehrung* und *Signaturmethode*. Bei der Volltextsuche werden mit Hilfe eines Teilstrings alle Objekte aufgespürt, die den String enthalten. Bei der Umkehrmethode, wird jedes Objekt als Liste von Indexbedingungen mit zugehörigen Gewichtungen. Jedes Objekt wird repräsentiert durch einen Vektor. Signaturtechniken lagern Objekte in Textordnern. Ein binärer String wird für jedes Textobjekt erzeugt.

Bilder und räumliche Objekte:

Die bisher verwendeten Methoden für die Erkennung und Speicherung dieser Objekte sind *pixelorientierte Annäherungen*, *Quad-Bäume*, *R-Bäume* oder *vektororientierte Annäherung*. Die meisten Techniken sind modellbasierend, wobei die Beschreibung auf bekannte Objekte (so genannte Modelle) zurückgeführt wird. Diese Modelle werden verwendet, um die Objekte zu identifizieren und die Position zu bestimmen.

Audio/Video:

Verschiedene Indextechniken basieren auf verschiedene Typen von *Suchschlüsseln*. Sie sind aber sehr von der Anfrage abhängig.

12.5 Multimediadatenbanken

Relationale Datenbanken wurden entwickelt, um herkömmliche Daten (integer, real, string etc.) zu verwalten. Das Archivieren von multimedialen Daten ist zwar oftmals möglich, jedoch merkt man beim Arbeiten mit den Multimediafunktionen an der Performance und Bedienbarkeit, dass die Hersteller mit ihren Multimedialösungen das Gesamtpaket einfach erweitern wollten, um »Randgruppen« zu ködern. Manchmal reicht es aber nicht aus, nur die dem multimedialen Objekt begleitenden Informationen bei einer Recherche zu verwenden. In einer Photodatenbank soll beispielsweise nach bestimmten Kennzeichen auf den Photos gesucht werden. Die multimedialen Daten müssen also in drei Bereiche unterteilt werden:

- Rohdaten,
- Beschreibungsdaten,
- Registrierungsdaten.

Eine Multimediadatenbank muss neben den formatierten Daten auch »unformatierte« Daten verwalten können. Es müssen schnelle Zugriffsmethoden für die neuen Datentypen bereitgestellt und die Hardwarearchitektur an die gesteigerten Leistungsanforderungen angepasst werden. Für Anwender, die eine Vielzahl von Mediadaten im Griff haben müssen, sind spezialisierte Datenbankprogramme unabdingbar. In den beiden folgenden Abschnitten sollen die Systeme *Cumulus* und *Media Center* als Repräsentanten der Multimediadatenbanken vorgestellt werden. *Cumulus* ist eine Profidatenbank der höheren Preisklasse und Media Center eine Alternative des niedrigeren Preissektors.

12.5.1 Cumulus

Cumulus ist eine komplexe Mediendatenbank aus dem Hause *Canto*. Das Produkt ist bereits seit einigen Jahren auf dem Markt und dementsprechend ausgereift. Mit Cumulus können Speichermedien oder Verzeichnisse nach bestimmten Dateitypen durchsucht und aus den Dateien *Vorschaubilder* (Thumbnails) extrahiert werden. Die Vorschaubilder werden in einem Katalog gespeichert und vermitteln einen groben Eindruck vom Bildinhalt. Mit den Thumbnails werden weitere Daten, wie Auflösung, Samplingfrequenz, Bitraten und Metadaten erfasst und archiviert. Zusätzlich können benutzerdefinierte Felder manuell gefüllt werden. Die Daten können als Informationen zur Datei und schließlich auch zur Suche oder zum Sortieren verwendet werden.

Mit den Entwicklerwerkzeugen von Cumulus können individuelle Multimedialösungen erstellt werden. Die Programmierumgebung für Windows basiert auf OLE Automation, so dass Sie Cumulus mit jeder Programmiersprache, die OLE Automation unterstützt, wie zum Beispiel Visual Basic, DELPHI und C++, steuern können.

Besonders nützlich sind auch die zahlreichen *PlugIns* (XPress, PageMaker, Illustrator etc.), die den Medienleuten die Weiterverarbeitung der Daten erleichtern. Außerdem bietet Canto weitere Optionen für Web-Anbindung oder das Zusammenarbeiten mit relationalen Datenbanken wie Oracle und MS SQL Server an.

12.5.2 Media Center Plus

Die Multimedia-Datenbank *Media Center Plus* vom Hersteller *Jasc* archiviert neben Bildern Animationen sowie Audio- und Videodateien. Unterstützt werden mehr als 50 Formate.

Media Center verwaltet die Dateien als *Thumbnails* in Alben, wobei es wie Canto Cumulus nicht die Originaldateien, sondern die *Referenzen* speichert. Der exakte Pfad zum Original wird mit jeder Referenzangabe abgelegt.

Ein Nachteil ist, dass man die Speicherinformationen der Referenz manuell korrigieren muss, wenn sich der Speicherort des Originals ändert.

Die Thumbnails lassen sich per Katalogdruck, als Slide-Show und als Webkatalog präsentieren. Es gibt Schnittstellen für Scanner und Digitalkameras. Zu den Stärken von Media Center gehören die vielseitigen *Sortier- und Druckmöglichkeiten* ebenso wie die gelungene Druckvorschau.

Media Center ist eine preisgünstige Alternative für den Einsteiger, die jedoch beim täglichen Arbeiten mit multimedialen Inhalten Mängel aufweist.

A Glossar

Abfrage Eine Abfrage ist eine spezielle Ansicht von Tabellen. In einer Abfrage selbst sind keine Daten gespeichert, sondern sie holt sich Werte aus einer oder mehreren Tabellen. Mit einer Abfrage können Daten aus (verschiedenen) Tabellen und Abfragen verknüpft, ausgewählt, gruppiert, gelöscht, geändert, erzeugt ... werden.

Account Ein Benutzerkonto für die Berechtigung, einen bestimmten Dienst ausführen zu dürfen. Im Internet ein Konto bei einem Provider oder einem Dienst. Über einen User-Namen und ein Passwort hat man Zugang zum Internet. So kann auch der Zugriff auf bestimmte Inhalte nur einem bestimmten Personenkreis erlaubt werden.

ActiveX Die ActiveX-Technologie ist eine Produktfamilie, die von Microsoft und anderen Firmen entwickelt wurde, um Web-Seiten Multimedia-Möglichkeiten wie Video, Audio, Animation und Virtuelle Realitäten hinzuzufügen. Dazu verwenden die Entwickler so genannte ActiveX-Controls zum Verbinden und Einbetten von Objekten »Object Linking and Embedding« (OLE).

Administrator Die Person, die ein Netzwerk oder eine Datenbank organisiert, aufrechterhält, Fehler beseitigt und es allgemein beaufsichtigt.

Aggregat Mehrgliedriger mathematischer Ausdruck, dessen einzelne Glieder durch + oder – miteinander verknüpft sind.

Algorithmus Eine Rechenvorschrift oder ein Verfahren, um z.B. Daten zu ver- bzw. entschlüsseln oder durch eine Iteration (Annäherung) die Wurzel einer Zahl zu bestimmen.

Alias Ein Name, üblicherweise kurz und einfach zu merken, der in einen anderen Namen übersetzt werden kann, der oft lang und schwer zu merken ist.

ANSI Das American National Standards Institute ist das amerikanische Normungsinstitut, ähnlich dem deutschen DIN. Vergleiche auch ISO. Siehe auch ASCII.

API Das Application Programming Interface ist eine dokumentierte Software-Schnittstelle, mit deren Hilfe ein Programm bestimmte Funktionen eines anderen Programms nutzen kann.

ASCII American Standard Code for Information Interchange ist ein grundlegendes Textformat, das die meisten Computer lesen können.

ASP 1.) Active Server Pages ist ein Standard der Firma Microsoft, um dynamische Seiten zu generieren. Dabei werden Scripts auf dem Web-Server (serverseitiges Scripting) ausgeführt. Der Anwender erkennt solche Seiten an Dokumenten mit der Endung »*.asp«. 2.) Application-Service-Provider bzw. Application-Service-Providing Anbieter von Programmen (Anwendungen, Applikationen), die auf einem Server ablaufen. Der Anwender muss dafür keine zusätzliche Software auf seinem PC installieren.

AVI steht für Audio Video Interleaved. Ein Microsoft-Standard für Audio- und Videodaten.

Backup Ein Backup ist eine technische Einrichtung oder eine Maßnahme, auf die im Notfall zurückgegriffen werden kann. Dies kann z.B. eine Bandkopie (Streamer) der Festplatte sein, die in regelmäßigen Abständen erstellt werden sollte.

Bandbreite Als Bandbreite bezeichnen Experten die Datenmenge, die sich innerhalb eines bestimmten Zeitraums übertragen lässt. Sie wird in Bit oder Millionen Bit pro Sekunde (bps oder Mbps) angegeben.

Batch Ein Satz von SQL-Anweisungen, die gemeinsam gesendet und als Gruppe ausgeführt werden. Bei einem Skript handelt es sich häufig um eine Folge von Batches, die nacheinander gesendet werden.

Benchmarking Darunter versteht man das kontinuierliche Vergleichen und Bewerten. Das können Unternehmen, Geschäftsprozesse sein, aber auch technische Geräte. Zuvor stellt man Regeln auf, wie und mit welchen Kriterien man etwas bewertet.

Benutzerkennung Der Name, mit dem der Benutzer sich dem Computersystem gegenüber identifiziert.

Beziehung repräsentiert den Bezug zwischen zwei Entitäten.

BLOB Ein Binary Large Object (großes binäres Objekt). Große Binär-Dateien fallen im Internet z.B. beim Download von Programm-, Video- oder Audiodaten an.

Boolesch (Boolean) Eine Operation oder ein Ausdruck, die bzw. der nur den Wert TRUE (wahr) oder FALSE (falsch) zurückgibt.

BSI Bundesamt für Sicherheit in der Informationstechnik.

B-Tree Index Erzeugt einen Indexbaum mit einer Liste von Zeilen-ID, die auf den Blättern sitzen. Benötigt mehr Speicher als der Bitmap-Index und ist der Standardindex für die meisten Datenbanken.

CERT Das Computer Emergency Response Team kümmert sich um die Datensicherheit im Internet.

Client Wenn Sie als Anwender mit einem anderen Rechner verbunden sind und dessen Dienste in Anspruch nehmen, besteht eine klassische Client/Server Beziehung.

COM Das Component Object Model ist ein Microsoft-Software-Standard zur Kommunikation zwischen Prozessen und Programmen. Mit Hilfe dieser objektorientierten Schnittstelle können Software-Produkte auch unterschiedlicher Hersteller untereinander Daten austauschen.

Compiler sind Übersetzungsprogramme, die den Quell- bzw. Sourcecode eines Programms einer höheren Programmiersprache (zusammen mit dem Linker) in den vom Prozessor ausführbaren Maschinencode umwandeln.

Cube Eine Menge von Daten, die in einer multidimensionalen Struktur organisiert und zusammengefasst und durch Dimensionen und Measures definiert sind.

Data Mining Eine Methode, um Zusammenhänge zwischen Datenbeständen zu finden und in eine sinnvolle Beziehung zu bringen. Durch Entdeckungsverfahren in großen Datenbeständen wird eine Entscheidungsunterstützung z.B. bei Verkaufsstrategien herbeigeführt.

Datawarehouse Ein »Datawarehouse« eines Unternehmens ist ein Zwischenlager für Daten unterschiedlichster Art. Auf diesen Datenpool können mehrere unterschiedliche Programme - je nach Aufgabenstellung (Bestellung, Fertigung, Lieferung, Rechnung, Kundenschreiben, etc.) - zugreifen. Redundante Daten können so vermieden werden. Die Datenhaltung erfolgt objektorientiert und nicht applikationsorientiert.

Datensatz Ein Datensatz ist ein Eintrag in eine Datenbank. Wenn man die Datenbank als einen Karteikasten bezeichnet, entspricht ein Datensatz einer Karteikarte. Zudem besteht er aus einem Satz zusammengehörender Datenfelder. Zum Beispiel können Name, Adresse und Telefonnummer einen Datensatz bilden.

Datenschutz Gesetze zum Schutz vor Missbrauch personenbezogener Daten. Personen müssen darüber unterrichtet werden, wenn ihre Daten erstmals eingespeichert werden.

Deadlock Ein Deadlock ist eine Situation, bei der Prozesse wechselseitig darauf warten, dass der jeweils andere Prozess von ihnen benötigte Ressourcen freigibt.

Debugger Programm für die Fehlersuche in einem anderen Programm, das eine schrittweise Abarbeitung und ein Testen der einzelnen Befehle ermöglicht.

Defragmentierer/ Defragmentierung Ein Programm, das die »zerstückelten« Dateien auf einem Datenträger wieder zusammenführt und wieder als zusammenhängende Dateien abspeichert.

Dimension Ein strukturelles Attribut eines Cubes, das eine organisierte Hierarchie aus Kategorien (Ebenen) zum Beschreiben von Daten der Faktentabelle darstellt. Diese Kategorien beschreiben normalerweise eine ähnliche Menge aus Elementen, die der Benutzer als Grundlage einer Analyse verwenden will.

DIN Deutsches Institut für Normung.

DLL Dynamic Link Library-Bibliothek für dynamisches Verbinden. Eine Datei, die ausführbaren Code und Daten enthält, die beim Laden oder während der Ausführung an ein Programm gebunden werden und nicht während der Herstellung der Verbindung.

Domain Teil bzw. Ebene in einer Adresse, den Konventionen (Rechtsbestimmungen) des Domäne-Name-Systems (DNS) folgend.

Durchsatz Tatsächlich erreichte Transferrate bei der Datenübertragung im Internet.

Entität Singuläres, von anderen Entitäten eindeutig unterscheidbares Exemplar aus einer Menge gleichartiger Personen, Gegenstände oder Begriffe aus der realen oder der Vorstellungswelt.

Ergonomie Die Untersuchung der Beziehungen zwischen dem Menschen und seiner Tätigkeit, der Maschine und der Arbeitsumgebung, insbesondere die Anwendung anatomischer, physiologischer und psychologischer Kenntnisse darauf.

Fremdschlüssel Die Spalte oder Kombination aus mehreren Spalten, deren Werte mit dem Primärschlüssel oder eindeutigen Schlüssel in derselben oder einer anderen Tabelle übereinstimmen. Dies wird auch als verweisender Schlüssel bezeichnet.

Grid Bezeichnung für ein Gitternetz.

Hash Index Der Hashing-Algorithmus verwendet eine Ziffernfolge, um einen Textstring zu identizieren und diesen zu komprimieren. Benötigt mehr Zeit zum Aufbau und wird nur von wenigen Datenbanken unterstützt.

Index Ein Index hilft Tabellen schneller durchsuchen zu können.

ISO International Standards Organization.

Join Beziehung zwischen mehreren Tabellen über Felder mit demselben Schlüsselinhalt.

Kardinalität Anzahl der Zeilen in einer Tabelle.

Konsistenz Die »Richtigkeit« von Daten und ihre Beziehungen zu anderen Daten. Wenn eine Datenbank inkonsistent ist, dann können Daten aus der gleichen oder anderen Quellen nicht miteinander kombiniert werden ohne zu falschen Ergebnissen zu führen.

Measure In einem Cube eine Menge von Werten, die auf einer Spalte in der Faktentabelle des Cubes beruhen und in der Regel numerisch sind. Measures sind die zentralen Werte, die aggregiert und analysiert werden.

Normalisierung Der Prozess der Aufteilung von großen redundanzbehafteten Tabellen in kleine ohne Redundanz.

Nullwert Nullwert beinhaltet nicht die Ziffer 0, sondern ist leer (auch keine Leerzeichen).

Objekt Ein Objekt ist eine Software-Struktur, die sowohl Daten als auch die dazugehörigen Methoden zusammenfasst.

ODBC Open Database Connectivity. Von Microsoft entwickelte Software-Schnittstelle, mit der Clients auf SQL-Datenbanken zugreifen können.

OLAP steht für »OnLine Analytical Processing« (»Datenbank mit analytischer Onlineverarbeitung«). Damit können umfangreiche (relationale) Datenbanksysteme abgefragt und ausgewertet werden, wie es z.B. in Unternehmen für das Controlling, Management, Rechnungswesen und Vertrieb erforderlich ist.

OLE Object Linking (verknüpfen) and Embedding (einbetten). Microsoft-Software-Standard, um Daten anderer Programme in ein Dokument integrieren zu können. Wird z.B. bei ActiveX verwendet, um Audio, Animationen, Video und Virtuelle Realitäten in Web-Seiten einzufügen.

OLTP steht für »OnLine Transaction Processing«. Erfolgen Änderungen in Datenbanken, stehen diese für den nächsten Zugriff aktualisiert zur Verfügung. OLTP-fähige Datenbanksysteme sind wichtig für Bankgeschäfte, Bestell- und Lagerhaltungssyteme.

OMG Object Management Group. Ein internationales Konsortium von Unternehmen aus verschiedenen Branchen mit dem Ziel, die Portabilität (Austausch, Übernahme), Interoperabilität (Zusammenarbeit), Verteilung und Wiederverwendung objektorientierter Software in heterogenen Umgebungen (also plattformunabhängig) zu fördern. Dazu wurden Standards wie OMA oder CORBA definiert.

Open-Source Software, die sowohl als Quelltext (source code) als auch in ausführbarer Form inspiziert, verändert werden darf.

Primärschlüssel Feld/Attribut in einer Tabelle mit den folgenden Eigenschaften:

- hohe Wahrscheinlichkeit der Einzigartigkeit (Uniqueness)
- muss immer einen Wert besitzen.

Recovery Unter Recovery versteht man alle Aktionen eines DBMS, die im Zusammenhang mit der Sicherung der Konsistenz der Datenbank beim Auftreten eines Fehlers stehen.

Redundanz Eine Information ist dann redundant, wenn sie ohne Informationsverlust weggelassen werden kann.

Referenzielle Integrität Besitzt eine Tabelle einen Fremdschlüssel, der auf einen Primärschlüssel einer referenzierten Tabelle zeigt, muss jeder Wert im Fremdschlüssel einen enstprechenden Eintrag in der referenzierten Tabelle besitzen. Dann besteht referenzielle Integrität.

Replikation Ein Vorgang, bei dem Daten und Datenbankobjekte aus einer Datenbank in eine andere Datenbank kopiert und verteilt werden. Anschließend werden diese Datenbanken aus Konsistenzgründen synchronisiert.

Rollback Von einem Rollback spricht man, wenn das DBMS im laufenden Betrieb aufgrund eines Fehlers eine bestimmte Transaktion zurücksetzen muss.

SQL Structured Query Language - strukturierte Abfragesprache.

Tabellenscan Eine Operation zum Abrufen von Daten, bei der das Datenbankmodul alle Seiten in einer Tabelle lesen muss, um die Zeilen zu finden, die für eine Abfrage gekennzeichnet sind.

Temporäre Tabelle Eine Tabelle, die in der temporären Datenbank platziert und am Ende der Sitzung gelöscht wird.

Thread Eine Komponente des Betriebssystems, die es ermöglicht, dass Anwendungen für mehrere Benutzer als verschiedene getrennte asynchrone Ausführungspfade ausgeführt werden.

Transaktion Eine Gruppe von Datenbankoperationen, die zu einer logischen Arbeitseinheit gruppiert sind und für die als Ganzes entweder ein Commit oder ein Rollback ausgeführt wird. Eine Transaktion ist atomar, konsistent, isoliert und beständig.

Trigger Eine gespeicherte Prozedur, die ausgeführt wird, wenn Daten in einer angegebenen Tabelle verändert werden. Trigger werden oftmals erstellt, um die referenzielle Integrität oder die Konsistenz logisch verbundener Daten in unterschiedlichen Tabellen zu erzwingen.

Trojanisches Pferd Ein Programm, das neben seiner eigentlichen Funktion noch eine zweite Funktion hat von der der Benutzer nichts weiß (beispielsweise Passwörter via E-Mail ins Internet verschicken).

Tupel Eine geordnete Auflistung aus Elementen verschiedener Dimensionen.

Union kombiniert mehrere Tabellen/Abfragen mit gleicher Struktur, ohne sie zu verknüpfen. Die Reihenfolge und Anzahl der Felder in jeder Tabelle/Abfragen muss gleich sein.

View Ein View (Sicht) ist eine spezielle Ansicht von Tabellen. In einer Sicht selbst sind keine Daten gespeichert, sondern sie holt sich Werte aus einer oder mehreren Tabellen. Mit einer Sicht können Daten aus (verschiedenen) Tabellen und Abfragen verknüpft, ausgewählt, gruppiert, gelöscht, geändert, erzeugt ... werden.

B SQL-Kurzreferenz

B.1 PostgreSQL

Name	**Syntax**	**Beispiel**
ABORT	ABORT [WORK | TRANSACTION]	ABORT WORK;
ALTER GROUP	ALTER GROUP *name* ADD USER *username* [, ...]	ALTER GROUP staff ADD USER karl, john
ALTER TABLE	ALTER TABLE [ONLY] *table* [*] ADD [COLUMN] *column type* ALTER TABLE [ONLY] *table* [*] ALTER [COLUMN] *column* { SET DEFAULT *value* | DROP DEFAULT } ALTER TABLE *table* [*] RENAME [COLUMN] *column* TO *newcolumn* ALTER TABLE *table* RENAME TO *newtable* ALTER TABLE *table* ADD *table constraint definition* ALTER TABLE *table* OWNER TO *new owner*	ALTER TABLE distributors ADD COLUMN address VARCHAR(30);
ALTER USER	ALTER USER *username* [WITH PASSWORD '*password*'] [CREATEDB | NOCREATEDB] [CREATEUSER | NOCREATEUSER] [VALID UNTIL '*abstime*']	ALTER USER kalli WITH PASSWORD 'hu8jmn3';
BEGIN	BEGIN [WORK | TRANSACTION]	BEGIN WORK;
CHECKPOINT	CHECKPOINT	
CLOSE	CLOSE *cursor*	CLOSE liahona;
CLUSTER	CLUSTER *indexname* ON *tablename*	CLUSTER emp_ind ON emp;

Name	Syntax	Beispiel
COMMENT	COMMENT ON [[DATABASE | INDEX | RULE | SEQUENCE | TABLE | TYPE | VIEW] *object_name* | COLUMN *table_name.column_name*| AGGREGATE *agg_name agg_type*| FUNCTION *func_name* (*arg1*, *arg2*, ...)| OPERATOR *op* (*leftoperand_type rightoperand_type*) | TRIGGER *trigger_name* ON *table_name*] IS '*text*'	COMMENT ON mytable IS 'This is my table.';
COMMIT	COMMIT [WORK | TRANSACTION]	COMMIT WORK;
COPY	COPY [BINARY] *table* [WITH OIDS] FROM { '*filename*' | stdin } [[USING] DELIMITERS '*delimiter*'] [WITH NULL AS '*null string*']	COPY country TO stdout USING DELIMITERS '|';
CREATE AGGREGATE	CREATE AGGREGATE *name* (BASETYPE = *input_data_type*, SFUNC = *sfunc*, STYPE = *state_type* [, FINALFUNC = *ffunc*] [, INITCOND = *initial_condition*])	
CREATE CONSTRAINT TRIGGER	CREATE CONSTRAINT TRIGGER *name* AFTER *events* ON *relation constraint attributes* FOR EACH ROW EXECUTE PROCEDURE *func* '(' *args* ')'	
CREATE DATABASE	CREATE DATABASE *name* [WITH [LOCATION = '*dbpath*'] [TEMPLATE = *template*] [ENCODING = *encoding*]]	CREATE DATABASE sample;
CREATE FUNCTION	CREATE FUNCTION *name* ([*ftype* [, ...]]) RETURNS *rtype* AS *definition* LANGUAGE '*langname*' [WITH (*attribute* [, ...])]	CREATE FUNCTION one() RETURNS int4 AS 'SELECT 1 AS RESULT' LANGUAGE 'sql';
CREATE GROUP	CREATE GROUP *name* [WITH [SYSID *gid*] [USER *username* [, ...]]]	CREATE GROUP edv;
CREATE INDEX	CREATE [UNIQUE] INDEX *index_name* ON *table* [USING *acc_name*] (*column* [*ops_name*] [, ...])	CREATE UNIQUE INDEX title_idx ON films (title);

Name	Syntax	Beispiel
CREATE RULE	CREATE RULE *name* AS ON *event* TO *object* [WHERE *condition*] DO [INSTEAD] *action*	CREATE RULE rulename AS ON SELECT TO emp DO INSTEAD
CREATE SEQUENCE	CREATE SEQUENCE *seqname* [INCREMENT *increment*] [MINVALUE *minvalue*] [MAXVALUE *maxvalue*] [START *start*] [CACHE *cache*] [CYCLE]	CREATE SEQUENCE serial START 101;
CREATE TABLE	CREATE [TEMPORARY | TEMP] TABLE *table_name* ({ *column_name type* [*column_constraint* [...]] | *table_constraint* } [, ...]) [INHERITS (*inherited_table* [, ...])] where *column_constraint* can be: [CONSTRAINT *constraint_name*] { NOT NULL | NULL | UNIQUE | PRIMARY KEY | DEFAULT *value* | CHECK (*condition*) | REFERENCES *table* [(*column*)] [MATCH FULL | MATCH PARTIAL] [ON DELETE *action*] [ON UPDATE *action*] [DEFERRABLE | NOT DEFERRABLE] [INITIALLY DEFERRED | INITIALLY IMMEDIATE] } and *table_constraint* can be: [CONSTRAINT *constraint_name*] { UNIQUE (*column_name* [, ...]) | PRIMARY KEY (*column_name* [, ...]) | CHECK (*condition*) | FOREIGN KEY (*column_name* [, ...]) REFERENCES *table* [(*column* [, ...])] [MATCH FULL | MATCH PARTIAL] [ON DELETE *action*] [ON UPDATE *action*] [DEFERRABLE | NOT DEFERRABLE] [INITIALLY DEFERRED | INITIALLY IMMEDIATE] }	CREATE TABLE distributors (Name VARCHAR(40) DEFAULT 'luso films', did INTEGER DEFAULT NEXTVAL('distributors_serial'), modtime TIMESTAMP DEFAULT now());
CREATE TABLE AS	CREATE [TEMPORARY | TEMP] TABLE *table* [(*column* [, ...])] AS *select_clause*	

Name	Syntax	Beispiel
CREATE TRIGGER	CREATE TRIGGER *name* { BEFORE | AFTER } { *event* [OR ...] } ON *table* FOR EACH { ROW | STATEMENT } EXECUTE PROCEDURE *func* (*arguments*)	CREATE TRIGGER if_dist_exists BEFORE INSERT OR UPDATE ON films FOR EACH ROW EXECUTE PROCEDURE check_primary_key ('did', 'distributors', 'did');
CREATE TYPE	CREATE TYPE *typename* (INPUT = *input_function*, OUTPUT = *output_function* , INTERNALLENGTH = { *internallength* | VARIABLE } [, EXTERNALLENGTH = { *externallength* | VARIABLE }] [, DEFAULT = "*default*"] [, ELEMENT = *element*] [, DELIMITER = *delimiter*] [, SEND = *send_function*] [, RECEIVE = *receive_function*] [, PASSEDBYVALUE] [, ALIGNMENT = *alignment*] [, STORAGE = *storage*])	CREATE TYPE box (INTERNALLENGTH = 8, INPUT = my_procedure_1, OUTPUT = my_procedure_2); CREATE TABLE myboxes (id INT4, description box);
CREATE USER	CREATE USER *username* [WITH [SYSID *uid*] [PASSWORD '*password*']] [CREATEDB | NOCREATEDB] [CREATEUSER | NOCREATEUSER] [IN GROUP *groupname* [, ...]] [VALID UNTIL '*abstime*']	CREATE USER jonathan;
CREATE VIEW	CREATE VIEW *view* AS SELECT *query*	CREATE VIEW kinds AS SELECT * FROM films WHERE kind = 'Comedy';
DECLARE	DECLARE *cursorname* [BINARY] [INSENSITIVE] [SCROLL] CURSOR FOR *query* [FOR { READ ONLY | UPDATE [OF *column* [, ...]]]	DECLARE liahona CURSOR FOR SELECT * FROM films;
DELETE	DELETE FROM [ONLY] *table* [WHERE *condition*]	DELETE FROM films WHERE kind <> 'Musical'; SELECT * FROM films;
DROP AGGREGATE	DROP AGGREGATE *name type*	DROP AGGREGATE myavg int4;

Name	Syntax	Beispiel
DROP DATABASE	DROP DATABASE *name*	
DROP FUNCTION	DROP FUNCTION *name* ([*type* [, ...]])	DROP FUNCTION sqrt(int4);
DROP GROUP	DROP GROUP *name*	DROP GROUP staff;
DROP INDEX	DROP INDEX *index_name* [, ...]	DROP INDEX title_idx;
DROP LANGUAGE	DROP [PROCEDURAL] LANGUAGE '*name*'	DROP PROCEDURAL LANGUAGE 'plsample';
DROP OPERATOR	DROP OPERATOR *id* (*lefttype* | NONE , *righttype* | NONE)	DROP OPERATOR ^ (int4, int4);
DROP RULE	DROP RULE *name* [, ...]	DROP RULE newrule;
DROP SEQUENCE	DROP SEQUENCE *name* [, ...]	DROP SEQUENCE serial;
DROP TABLE	DROP TABLE *name* [, ...]	DROP TABLE films, distributors;
DROP TRIGGER	DROP TRIGGER *name* ON *table*	DROP TRIGGER if_dist_exists ON films;
DROP TYPE	DROP TYPE *typename* [, ...]	DROP TYPE box;
DROP USER	DROP USER *name*	DROP USER jonathan;
DROP VIEW	DROP VIEW *name* [, ...]	DROP VIEW kinds;
END	END [WORK | TRANSACTION]	END WORK;
EXPLAIN	EXPLAIN [VERBOSE] *query*	EXPLAIN SELECT * FROM foo; NOTICE: QUERY PLAN: Seq Scan on foo (cost=0.00..2.28 rows=128 width=4) EXPLAIN
FETCH	FETCH [*direction*] [*count*] { IN | FROM } *cursor*	FETCH FORWARD 5 IN liahona;
GRANT	GRANT *privilege* [, ...] ON *object* [, ...] TO { PUBLIC | GROUP *group* | *username* }	GRANT INSERT ON films TO PUBLIC;
INSERT	INSERT INTO *table* [(*column* [, ...])] { DEFAULT VALUES | VALUES (*expression* [, ...]) | SELECT *query* }	INSERT INTO films VALUES ('UA502','Bananas',105,'1971-07-13','Comedy',INTERVAL '82 minute');
LISTEN	LISTEN *name*	LISTEN virtual; NOTIFY virtual;
LOAD	LOAD '*filename*'	LOAD '/usr/postgres/demo/circle.o'

Name	Syntax	Beispiel
LOCK	LOCK [TABLE] *name*	BEGIN WORK; LOCK TABLE films IN SHARE MODE; SELECT id FROM films WHERE name = 'Star Wars: Episode I - The Phantom Menace'; -- Do ROLLBACK if record was not returned INSERT INTO films_user_comments VALUES (_id_, 'GREAT! I was waiting for it for so long!'); COMMIT WORK;
MOVE	MOVE [*direction*] [*count*] { IN | FROM } *cursor*	MOVE FORWARD 5 IN liahona;
NOTIFY	NOTIFY *name*	LISTEN virtual; NOTIFY virtual; Asynchronous NOTIFY 'virtual' from backend with pid '8448' received.
REINDEX	REINDEX { TABLE | DATABASE | INDEX } *name* [FORCE]	REINDEX TABLE mytable;
REVOKE	REVOKE *privilege* [, ...] ON *object* [, ...] FROM { PUBLIC | GROUP *groupname* | *username* }	REVOKE INSERT ON films FROM PUBLIC;
RESET	RESET *variable*	RESET DateStyle;
ROLLBACK	ROLLBACK [WORK | TRANSACTION]	ROLLBACK WORK;
SELECT	SELECT [ALL | DISTINCT [ON (*expression* [, ...])]] * | *expression* [AS *output_name*] [, ...] [FROM *from_item* [, ...]] [WHERE *condition*] [GROUP BY *expression* [, ...]] [HAVING *condition* [, ...]] [{ UNION | INTERSECT | EXCEPT [ALL] } *select*] [ORDER BY *expression* [ASC | DESC | USING *operator*] [, ...]] [FOR UPDATE [OF *tablename* [, ...]]] [LIMIT { *count* | ALL } [{ OFFSET | , } *start*]]	SELECT f.title, f.did, d.name, f.date_prod, f.kind FROM distributors d, films f WHERE f.did = d.did

Name	Syntax	Beispiel
SELECT INTO	SELECT [ALL | DISTINCT [ON (*expression* [, ...])]] * | *expression* [AS *output_name*] [, ...] INTO [TEMPORARY | TEMP] [TABLE] *new_table* [FROM *from_item* [, ...]] [WHERE *condition*] [GROUP BY *expression* [, ...]] [HAVING *condition* [, ...]] [{ UNION | INTERSECT | EXCEPT [ALL] } *select*] [ORDER BY *expression* [ASC | DESC | USING *operator*] [, ...]] [FOR UPDATE [OF *tablename* [, ...]]] [LIMIT { *count* | ALL } [{ OFFSET | , } *start*]]	
SET	SET *variable* { TO | = } { *value* | '*value*' | DEFAULT } SET TIME ZONE { '*timezone*' | LOCAL | DEFAULT }	SET DATESTYLE TO Postgres, European;
SET CONSTRAINTS	SET CONSTRAINTS { ALL | *constraint* [, ...] } { DEFERRED | IMMEDIATE }	
SET TRANSACTION	SET TRANSACTION ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE } SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE }	
SHOW	SHOW *name*	SHOW DateStyle;
TRUNCATE	TRUNCATE [TABLE] *name*	TRUNCATE TABLE bigtable;
UNLISTEN	UNLISTEN { *notifyname* | * }	LISTEN virtual; LISTEN NOTIFY virtual; NOTIFY Asynchronous NOTIFY 'virtual' from backend with pid '8448' received
UPDATE	UPDATE [ONLY] *table* SET *col* = *expression* [, ...] [FROM *fromlist*] [WHERE *condition*]	UPDATE films SET kind = 'Dramatic' WHERE kind = 'Drama'; SELECT * FROM films WHERE kind = 'Dramatic' OR kind = 'Drama';
VACUUM	VACUUM [VERBOSE] [ANALYZE] [*table*]	

B.2 Transact-SQL

B.2.1 Aggregatfunktionen

Funktion	Syntax	Beispiel
AVG	AVG ([ALL | DISTINCT] expression)	SELECT AVG(advance), SUM(sales) FROM titles WHERE type = 'business'
BINARY_CHECKSUM	BINARY_CHECKSUM (* | expression [,...n])	CREATE TABLE TableBC (ProductID int, bchecksum int) INSERT INTO TableBC SELECT ProductID, BINARY_CHECKSUM(*) FROM Products
CHECKSUM	CHECKSUM (* | expression [,...n])	ALTER TABLE Products ADD cs_Pname AS checksum(product) CREATE INDEX Pname_index ON Products (cs_Pname);
CHECKSUM_AGG	CHECKSUM_AGG ([ALL | DISTINCT] expression)	SELECT CHECKSUM_AGG(BINARY_ CHECKSUM(*)) FROM Products;
COUNT	COUNT ({ [ALL | DISTINCT] expression] | * })	SELECT COUNT(DISTINCT city) FROM authors;
COUNT_BIG	COUNT_BIG ({ [ALL | DISTINCT] expression } | *)	
GROUPING	GROUPING (column_name)	SELECT royalty, SUM(advance) 'total advance', GROUPING(royalty) 'grp' FROM titles GROUP BY royalty WITH ROLLUP;
MAX	MAX ([ALL | DISTINCT] expression)	SELECT MAX(ytd_sales) FROM titles;
MIN	MIN ([ALL | DISTINCT] expression)	SELECT min(ytd_sales) FROM titles;
SUM	SUM ([ALL | DISTINCT] expression)	SELECT type, SUM(price), SUM(advance) FROM titles WHERE type LIKE '%cook' GROUP BY type ORDER BY type;

Funktion	Syntax	Beispiel
STDEV	STDEV (expression)	SELECT STDEV(royalty) FROM titles;
STDEVP	STDEVP (expression)	SELECT STDEVP(royalty) FROM titles;
VAR	VAR (expression)	SELECT VAR(royalty) FROM titles;
VARP	VARP (expression)	SELECT VARP(royalty); FROM titles

B.2.2 Konfigurationsfunktionen

Funktion	Syntax	Beispiel
@@DATEFIRST	@@DATEFIRST	SET DATEFIRST 5 SELECT @@DATEFIRST AS '1st Day', DATEPART(dw, GETDATE()) AS 'Today'
@@DBTS	@@DBTS	SELECT @@DBTS
@@LANGID	@@LANGID	SET LANGUAGE 'Italian' SELECT @@LANGID AS 'Language ID'
@@LANGUAGE	@@LANGUAGE	SELECT @@LANGUAGE AS 'Language Name'
@@LOCK_TIMEOUT	@@LOCK_TIMEOUT	SELECT @@LOCK_TIMEOUT
@@MAX_CONNECTIONS	@@MAX_CONNECTIONS	SELECT @@MAX_CONNECTIONS
@@MAX_PRECISION	@@MAX_PRECISION	SELECT @@MAX_PRECISION
@@NESTLEVEL	@@NESTLEVEL	CREATE PROCEDURE innerproc AS SELECT @@NESTLEVEL AS 'Inner Level'
@@OPTIONS	@@OPTIONS	SET NOCOUNT ON IF @@OPTIONS & 512 > 0 RAISERROR ('Current user has SET NOCOUNT turned on.',1,1)
@@REMSERVER	@@REMSERVER	CREATE PROCEDURE chk_serv AS SELECT @@REMSERVER
@@SERVICENAME	@@SERVICENAME	SELECT @@SERVICENAME
@@SERVERNAME	@@SERVERNAME	SELECT @@SERVERNAME

Funktion	Syntax	Beispiel
@@SPID	@@SPID	SELECT @@SPID AS 'ID', SYSTEM_USER AS 'Login Name', USER AS 'User Name'
@@TEXTSIZE	@@TEXTSIZE	SELECT @@TEXTSIZE SET TEXTSIZE 2048 SELECT @@TEXTSIZE
@@VERSION	@@VERSION	SELECT @@VERSION

B.2.3 Datumsfunktionen

Funktion	Syntax	Beispiel
DATEADD	DATEADD (datepart, number, date)	SELECT DATEADD(day, 21, pubdate) AS timeframe FROM titles
DATEDIFF	DATEDIFF (datepart, startdate, enddate)	SELECT DATEDIFF(day, pubdate, getdate()) AS no_of_days FROM titles
DATENAME	DATENAME (datepart, date)	SELECT DATENAME(month, getdate()) AS 'Month Name'
DATEPART	DATEPART (datepart, date)	SELECT GETDATE() AS 'Current Date'
DAY	DAY (date)	SELECT DAY('03/12/1998') AS 'Day Number'
GETDATE	GETDATE ( )	SELECT GETDATE()
GETUTCDATE	GETUTCDATE()	
MONTH	MONTH (date)	SELECT "Month Number" = MONTH('03/12/1998')
YEAR	YEAR (date)	SELECT "Year Number" = YEAR('03/12/1998')

B.2.4 Mathematische Funktionen

Funktion	Syntax	Beispiel
ABS	ABS (numeric_expression)	SELECT ABS(-1.0), ABS(0.0), ABS(1.0)

Funktion	Syntax	Beispiel
ACOS	ACOS (float_expression)	SET NOCOUNT OFF DECLARE @angle float SET @angle = -1 SELECT 'The ACOS of the angle is:' + CONVERT (varchar, ACOS(@angle))
ASIN	ASIN (float_expression)	DECLARE @angle float SET @angle = -1.01 SELECT 'The ASIN of the angle is: ' + CONVERT (varchar, ASIN(@angle));
ATAN	ATAN (float_expression)	SELECT 'The ATAN of -45.01 is: ' + CONVERT (varchar, ATAN (-45.01))
ATN2	ATN2 (float_expression, float_expression)	ATN2 (float_expr, float_expr)
CEILING	CEILING (numeric_expression)	SELECT CEILING($123.45), CEILING($-123.45), CEILING($0.0);
COS	COS (float_expression)	DECLARE @angle float SET @angle = 14.78 SELECT 'The COS of the angle is:' + CONVERT(varchar, COS(@angle))
COT	COT (float_expression)	DECLARE @angle float SET @angle = 124.1332 SELECT 'The COT of the angle is:' + CONVERT(varchar, COT(@angle))
DEGREES	DEGREES (numeric_expression)	SELECT 'The number of degrees in PI/2 radians is: ' + CONVERT(varchar, DEGREES((PI()/2)))
EXP	EXP (float_expression)	DECLARE @var float SET @var = 378.615345498 SELECT 'The EXP of the variable is:' + CONVERT(varchar, EXP(@var))
FLOOR	FLOOR (numeric_expression)	SELECT FLOOR(123.45), FLOOR(-123.45), FLOOR($123.45)
LOG	LOG (float_expression)	

Funktion	Syntax	Beispiel
LOG10	LOG10 (float_expression)	DECLARE @var float SET @var = 145.175643 SELECT 'The LOG10 of the variable is:' + CONVERT(varchar, LOG10(@var))
PI	PI ( )	SELECT PI();
POWER	POWER (numeric_expression, y)	SELECT POWER(2.0, -100.0);
RADIANS	RADIANS (numeric_expression)	SELECT RADIANS(1e-307);
RAND	RAND ([seed])	DECLARE @counter smallint SET @counter = 1 WHILE @counter < 5 BEGIN SELECT RAND(@counter) Random_Number SET NOCOUNT ON SET @counter = @counter + 1 SET NOCOUNT OFF END
ROUND	ROUND (numeric_expression, length [, function])	SELECT ROUND(123.9994, 3), ROUND(123.9995, 3);
SIGN	SIGN (numeric_expression)	DECLARE @value real SET @value = -1 WHILE @value < 2 BEGIN SELECT SIGN(@value) SET NOCOUNT ON SELECT @value = @value + 1 SET NOCOUNT OFF END SET NOCOUNT OFF
SIN	SIN (float_expression)	DECLARE @angle float SET @angle = 45.175643 SELECT 'The SIN of the angle is: ' + CONVERT(varchar, SIN(@angle))
SQUARE	SQUARE (float_expression)	DECLARE @h float, @r float SET @h = 5 SET @r = 1 SELECT PI()* SQUARE(@r)* @h AS 'Cyl Vol'

Funktion	Syntax	Beispiel
SQRT	SQRT (float_expression)	DECLARE @myvalue float SET @myvalue = 1.00 WHILE @myvalue < 10.00 BEGIN SELECT SQRT(@myvalue) SELECT @myvalue = @myvalue + 1 END
TAN	TAN (float_expression)	SELECT TAN(PI()/2)

B.2.5 Metadatenfunktionen

Funktion	Syntax	Beispiel
COL_LENGTH	COL_LENGTH ('table', 'column')	SELECT COL_LENGTH('t1','c1') AS 'VarChar', COL_LENGTH('t1','c2') AS 'NVarChar';
COL_NAME	COL_NAME (table_id, column_id)	SET NOCOUNT OFF SELECT COL_NAME (OBJECT_ID ('Employees'), 1);
COLUMNPROPERTY	COLUMNPROPERTY (id, column, property)	SELECT COLUMNPROPERTY(OBJECT_ID('authors'),'au_lname ','PRECISION')
DATABASEPROPERTY	DATABASEPROPERTY (database, property)	SELECT DATABASEPRO- PERTY('master', 'IsTruncLog')
DATABASEPROPERTYEX	DATABASEPROPERTYEX (database, property)	SELECT DATABASEPROPER- TYEX('Northwind', 'IsAuto- Shrink')
DB_ID	DB_ID (['database_name'])	SELECT name, DB_ID(name) AS DB_ID FROM sysdatabases ORDER BY dbid
DB_NAME	DB_NAME (database_id)	SELECT dbid, DB_NAME(dbid) AS DB_NAME FROM sysdatabases ORDER BY dbid
FILE_ID	FILE_ID ('file_name')	SELECT FILE_ID('master')
FILE_NAME	FILE_NAME (file_id)	SELECT FILE_NAME(1)
FILEGROUP_ID	FILEGROUP_ID ('filegroup_name')	SELECT FILEGROUP_ID('default')

Funktion	Syntax	Beispiel
FILEGROUP_NAME	FILEGROUP_NAME (filegroup_id)	SELECT FILEGROUP_NAME(1)
FILEGROUPPROPERTY	FILEGROUPPROPERTY (filegroup_name, property)	SELECT FILEGROUPPROPERTY('primary', 'IsUserDefinedFG')
FILEPROPERTY	FILEPROPERTY (file_name, property)	SELECT FILEPROPERTY ('master', 'IsPrimaryFile')
fn_listextendedproperty	fn_listextendedproperty ({ default | [@name =] 'property_name' | NULL } , { default | [@level0type =] 'level0_object_type' | NULL } , { default | [@level0name =] 'level0_object_name' | NULL } , { default | [@level1type =] 'level1_object_type' | NULL } , { default | [@level1name =] 'level1_object_name' | NULL } , { default | [@level2type =] 'level2_object_type' | NULL } , { default | [@level2name =] 'level2_object_name' | NULL })	SELECT * FROM ::fn_listextendedproperty(NULL, NULL, NULL, NULL, NULL, NULL, NULL) - Oder - SELECT * FROM ::fn_listextendedproperty(default , default, default, default, default, default, default)
FULLTEXTCATALOGPROPERTY	FULLTEXTCATALOGPROPERTY (catalog_name , property)	SELECT fulltextcatalogproperty('Cat_Desc', 'ItemCount')
FULLTEXTSERVICEPROPERTY	FULLTEXTSERVICEPROPERTY (property)	SELECT fulltextserviceproperty('IsFulltextInstalled')
INDEX_COL	INDEX_COL ('table', index_id, key_id)	
INDEXKEY_PROPERTY	INDEXKEY_PROPERTY (table_ID, index_ID, key_ID, property)	SELECT indexkey_property(OBJECT_ID ('authors'),2,2,'ColumnId') SELECT indexkey_property(OBJECT_ID ('authors'),2,2,'IsDescending')

Funktion	Syntax	Beispiel
INDEXPROPERTY	INDEXPROPERTY (table_ID, index, property)	SELECT INDEXPRO- PERTY(OBJECT_ID('authors'), 'UPKCL_auidind', 'IsPadIndex'
OBJECT_ID	OBJECT_ID ('object')	SELECT OBJECT_ID('pubs..authors')
OBJECT_NAME	OBJECT_NAME (object_id)	SELECT TABLE_CATALOG, TABLE_NAME FROM INFORMATION_SCHEMA. TABLES WHERE TABLE_NAME = OBJECT_NAME(1977058079)
OBJECTPROPERTY	OBJECTPROPERTY (id, property)	IF OBJECTPROPERTY (object_id('authors'), 'ISTABLE') = 1 print 'Authors is a table' ELSE IF OBJECTPROPERTY (object_id('authors'), 'ISTABLE') = 0 print 'Authors is not a table' ELSE IF OBJECTPROPERTY (object_id('authors'),'ISTABLE') IS NULL print 'ERROR: Authors is not an object'
@@PROCID	@@PROCID	CREATE PROCEDURE testpro- cedure AS SELECT @@PROCID AS 'ProcID' GO EXEC testprocedure

Funktion	Syntax	Beispiel
SQL_VARIANT_PROPERTY	SQL_VARIANT_PROPERTY (expression, property)	CREATE TABLE tableA (colA sql_variant, colB int) INSERT INTO tableA values (cast (46279.1 as decimal(8,2)), 1689) SELECT SQL_VARIANT_PROPERTY (colA,'BaseType'), SQL_VARIANT_PROPERTY (colA,'Precision'), SQL_VARIANT_PROPERTY (colA,'Scale') FROM tableA WHERE colB = 1689
TYPEPROPERTY	TYPEPROPERTY (type, property)	SELECT TYPEPROPERTY ('tinyint', 'PRECISION')

B.2.6 Sicherheitsfunktionen

Funktion	Syntax	Beispiel
fn_trace_geteventinfo	fn_trace_geteventinfo ([@traceid =] trace_id)	
fn_trace_getfilterinfo	fn_trace_getfilterinfo ([@traceid =] trace_id)	
fn_trace_getinfo	fn_trace_getinfo ([@traceid =] trace_id)	
fn_trace_gettable	fn_trace_gettable([@filename =] file- name, [@numfiles =] number_files)	SELECT * INTO temp_trc FROM ::fn_trace_gettable (c:\my_trace.trc", default)
HAS_DBACCESS	HAS_DBACCESS ('database_name')	

Funktion	Syntax	Beispiel

IS_MEMBER	IS_MEMBER ({ 'group' | 'role' })	IF IS_MEMBER ('db_owner') = 1
		  print 'Current user is a member of the db_owner role'
		ELSE IF IS_MEMBER ('db_owner') = 0
		  print 'Current user is NOT a member of the db_owner role'
		ELSE IF IS_MEMBER ('db_owner') IS NULL
		  print 'ERROR: Invalid group / role specified'

IS_SRVROLEMEMBER	IS_SRVROLEMEMBER ('role' [, 'login'])	IF IS_SRVROLEMEMBER ('sysadmin') = 1
		  print 'Current user''s login is a member of the sysadmin role'
		ELSE IF IS_SRVROLEMEMBER ('sysadmin') = 0
		  print 'Current user''s login is NOT a member of the sysadmin role'
		ELSE IF IS_SRVROLEMEMBER ('sysadmin') IS NULL
		  print 'ERROR: Invalid server role specified'

SUSER_SID	SUSER_SID (['login'])	SELECT SUSER_SID('sa')
SUSER_SNAME	SUSER_SNAME ([server_user_sid])	SELECT SUSER_SNAME(0x01)
USER_ID	USER_ID (['user']	USER_ID (['user']
USER	USER	DECLARE @usr char(30) SET @usr = user SELECT 'The current user's database username is: '+ @usr

B.2.7 Zeichenfolgenfunktionen

Funktion	Syntax	Beispiel

ASCII	ASCII (character_expression)	CHAR(ASCII(SUBSTRING(@string, @position, 1)))
		  SET @position = @position + 1
		  END
		SET NOCOUNT OFF

Funktion	Syntax	Beispiel
CHAR	CHAR (integer_expression)	SET @position = 1 SET @string = 'New Moon' WHILE @position <= DATALENGTH(@string) BEGIN SELECT ASCII(SUBST- RING(@string, @position, 1)), CHAR(ASCII(SUBST- RING(@string, @position, 1))) SET @position = @position + 1 END
CHARINDEX	CHARINDEX (expression1, expression2 [, start_location])	SELECT CHARINDEX ('won- derful', notes) FROM titles WHERE title_id = 'TC3218' ('wonderful', notes, 5) FROM titles WHERE title_id = 'TC3218'
DIFFERENCE	DIFFERENCE (character_expression, character_expression)	SELECT SOUNDEX('Green'), SOUNDEX('Greene'), DIFFE- RENCE('Green','Greene')
LEFT	LEFT (character_expression, integer_expression)	SELECT LEFT(title, 5) FROM titles ORDER BY title_id
LEN	LEN (string_expression)	SELECT LEN (CompanyName) AS 'Length', CompanyName FROM Customers WHERE Country = 'Finland'
LOWER	LOWER (character_expression)	SELECT LOWER(SUBST- RING(title, 1, 20)) AS Lower, UPPER(SUBSTRING (title, 1, 20)) AS Upper, LOWER(UPPER(SUBST- RING(title, 1, 20))) As LowerUpper FROM titles WHERE price between 11.00 and 20.00

Funktion	Syntax	Beispiel

LTRIM	LTRIM (character_expression)	DECLARE @string_to_trim varchar(60)
		SET @string_to_trim = ' Five spaces are at the beginning of this
		  string.'
		SELECT 'Here is the string without the leading spaces: ' +
		  LTRIM(@string_to_trim)

NCHAR	NCHAR (integer_expression)	DECLARE @nstring nchar(8) SET @nstring = N'København' SELECT UNICODE (SUBSTRING(@nstring, 2, 1)), NCHAR(UNICODE (SUBSTRING(@nstring, 2, 1)))
PATINDEX	PATINDEX ('%pattern%' , expression)	SELECT PATINDEX ('%wonderful%', notes) FROM titles WHERE title_id = 'TC3218'
REPLACE	REPLACE ('string_expression1' , 'string_expression2' , 'string_expression3')	SELECT REPLACE ('abcdefghicde','cde','xxx')
QUOTENAME	QUOTENAME('character_string' [, 'quote_character'])	SELECT QUOTE- NAME('abc[]def')
REPLICATE	REPLICATE(character_expression, integer_expression)	SELECT REPLICATE (au_fname, 2) FROM authors ORDER BY au_fname
REVERSE	REVERSE (character_expression)	SELECT REVERSE(au_fname) FROM authors ORDER BY au_fname
RIGHT	RIGHT (character_expression, integer_expression)	SELECT RIGHT(au_fname, 5) FROM authors ORDER BY au_fname
RTRIM	RTRIM (character_expression)	DECLARE @string_to_trim varchar(60) SET @string_to_trim = 'Four spaces are after the period in this sentence. ' SELECT 'Here is the string without the leading spaces: ' + CHAR(13) + RTRIM(@string_to_trim)

Funktion	Syntax	Beispiel
SOUNDEX	SOUNDEX (character_expression)	-- Using SOUNDEX SELECT SOUNDEX ('Smith'), SOUNDEX ('Smythe')
SPACE	SPACE (integer_expression)	SELECT RTRIM(au_lname) + ',' + SPACE(2) + LTRIM(au_fname) FROM authors ORDER BY au_lname, au_fname
STR	STR (float_expression [, length [, decimal]])	SELECT STR(123.45, 6, 1)
STUFF	STUFF (character_expression, start, length , character_expression)	SELECT STUFF('abcdef', 2, 3, 'ijklmn')
SUBSTRING	SUBSTRING (expression , start , length)	SELECT au_lname, SUBSTRING(au_fname, 1, 1) FROM authors ORDER BY au_lname
UNICODE	UNICODE ('ncharacter_expression')	DECLARE @nstring nchar(12) SET @nstring = N'Åkergatan 24' SELECT UNICODE(@nstring), NCHAR(UNICODE(@nstring))
UPPER	UPPER (character_expression)	SELECT UPPER(RTRIM(au_lname)) + ', ' + au_fname AS Name FROM authors ORDER BY au_lname

B.2.8 Systemfunktionen

Funktion	Syntax	Beispiel
APP_NAME	APP_NAME ( )	DECLARE @CurrentApp varchar(35) SET @CurrentApp = APP_NAME() IF @CurrentApp <> 'MS SQL Query Analyzer' PRINT 'This process was not started by a SQL Query Analyzer query session.'

Funktion	Syntax	Beispiel
CASE	CASE input_expression WHEN when_expression THEN result_expression [...n] [ELSE else_result_expression] END **Die komplexe CASE-Funktion:** CASE WHEN Boolean_expression THEN result_expression [...n] [ELSE else_result_expression] END	SELECT Category = CASE type WHEN 'popular_comp' THEN 'Popular Computing' WHEN 'mod_cook' THEN 'Modern Cooking' WHEN 'business' THEN 'Business' WHEN 'psychology' THEN 'Psychology' WHEN 'trad_cook' THEN 'Traditional Cooking' ELSE 'Not yet categorized' END, CAST(title AS varchar(25)) AS 'Shortened Title', price AS Price FROM titles WHERE price IS NOT NULL ORDER BY type, price COMPUTE AVG(price) BY type
CAST und CONVERT	CAST (expression AS data_type)	SELECT SUBSTRING(title, 1, 30) AS Title, ytd_sales FROM titles WHERE CAST(ytd_sales AS char(20)) LIKE '3%' SELECT SUBSTRING(title, 1, 30) AS Title, ytd_sales FROM titles WHERE CONVERT(char(20), ytd_sales) LIKE '3%'
COALESCE	COALESCE (expression [,...n])	
COLLATIONPROPERTY	COLLATIONPROPERTY (collation_name, property)	SELECT COLLATIONPROPERTY('Traditional_Spanish_CS_AS_KS_WS', 'CodePage')
CURRENT_TIMESTAMP	SELECT COLLATIONPROPERTY('Traditional_Spanish_CS_AS_KS_WS', 'CodePage')	SELECT 'The current time is: '+ CONVERT(char(30), CURRENT_TIMESTAMP)
CURRENT_USER	CURRENT_USER	SELECT 'The current user is: '+ convert(char(30), CURRENT_USER)

Funktion	Syntax	Beispiel
DATALENGTH	DATALENGTH (expression)	SELECT length = DATALENGTH(pub_name), pub_name FROM publishers ORDER BY pub_name
@@ERROR	@@ERROR	UPDATE authors SET au_id = '172 32 1176' WHERE au_id = "172-32-1176" IF @@ERROR = 547 print "A check constraint violation occurred"
fn_helpcollations	fn_helpcollations ()	
fn_servershareddrives	fn_servershareddrives()	SELECT * FROM ::fn_servershareddrives()
fn_virtualfilestats	fn_virtualfilestats ([@DatabaseID=] database_id , [@FileID =] file_id)	SELECT * FROM :: fn_virtualfilestats(1, 1)
FORMATMESSAGE	FORMATMESSAGE (msg_number, param_value [,...n])	DECLARE @var1 VARCHAR(100) SELECT @var1 = FORMATMESSAGE(50001, 'Table1', 5)
GETANSINULL	GETANSINULL (['database'])	SELECT GETANSINULL('pubs')
HOST_ID	HOST_ID ( )	CREATE TABLE Orders (OrderID INT PRIMARY KEY, CustomerID NCHAR(5) REFERENCES Customers(CustomerID), TerminalID CHAR(8) NOT NULL DEFAULT HOST_ID(), OrderDate DATETIME NOT NULL, ShipDate DATETIME NULL, ShipperID INT NULL REFERENCES Shippers (ShipperID))

Funktion	Syntax	Beispiel
HOST_NAME	HOST_NAME ( )	CREATE TABLE Orders (OrderID INT PRIMARY KEY, CustomerID NCHAR(5) REFERENCES Customers (CustomerID), Workstation NCHAR(30) NOT NULL DEFAULT HOST_NAME(), OrderDate DATETIME NOT NULL, ShipDate DATETIME NULL, ShipperID INT NULL REFERENCES Shippers (ShipperID))
IDENT_CURRENT	IDENT_CURRENT('table_name')	DROP TABLE t6 DROP TABLE t7 CREATE TABLE t6(id int IDENTITY) CREATE TABLE t7(id int IDENTITY(100,1)) CREATE TRIGGER t6ins ON t6 FOR INSERT AS BEGIN INSERT t7 DEFAULT VALUES END
IDENT_INCR	IDENT_INCR ('table_or_view')	SELECT TABLE_NAME, IDENT_INCR(TABLE_NAME) AS IDENT_INCR FROM INFORMATION_SCHEMA. TABLES WHERE IDENT_INCR(TABLE_NAME) IS NOT NULL

Funktion	Syntax	Beispiel
IDENT_SEED	IDENT_SEED ('table_or_view')	SELECT TABLE_NAME, IDENT_SEED(TABLE_NAME) AS IDENT_SEED FROM INFORMATION_SCHEMA. TABLES WHERE IDENT_SEED(TABLE_NAME) IS NOT NULL
@@IDENTITY	@@IDENTITY	INSERT INTO jobs (job_desc,min_lvl,max_lvl) VALUES ('Accountant',12,125) SELECT @@IDENTITY AS 'Identity'
IDENTITY (Funktion)	IDENTITY (data_type [, seed , increment]) AS column_name	IF EXISTS(SELECT TABLE_NAME FROM INFORMATION_SCHEMA. TABLES WHERE TABLE_NAME = 'employees') DROP TABLE employees EXEC sp_dboption 'pubs', 'select into/bulkcopy', 'true' SELECT emp_id AS emp_num, fname AS first, minit AS middle, lname AS last, IDENTITY(smallint, 100, 1) AS job_num, job_lvl AS job_level, pub_id, hire_date INTO employees FROM employee EXEC sp_dboption 'pubs', 'select into/bulkcopy', 'false'
ISDATE	ISDATE (expression)	DECLARE @datestring varchar(8) SET @datestring = '12/21/98' SELECT ISDATE(@datestring)
ISNULL	ISNULL (check_expression, replacement_value)	SELECT AVG(ISNULL(price, $10.00)) FROM titles

Funktion	Syntax	Beispiel
ISNUMERIC	ISNUMERIC (expression)	SELECT ISNUMERIC(zip) FROM authors
NEWID	NEWID ( )	-- Creating a local variable with DECLARE/SET syntax. DECLARE @myid uniqueidentifier SET @myid = NEWID() PRINT 'Value of @myid is: '+ CONVERT(varchar(255), @myid)
NULLIF	NULLIF (expression , expression)	IF EXISTS (SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME = 'budgets') DROP TABLE budgets SET NOCOUNT ON CREATE TABLE budgets (dept tinyint IDENTITY, current_year decimal NULL, previous_year decimal NULL) INSERT budgets VALUES(100000, 150000) INSERT budgets VALUES(NULL, 300000) INSERT budgets VALUES(0, 100000) INSERT budgets VALUES(NULL, 150000) INSERT budgets VALUES(300000, 250000) SET NOCOUNT OFF SELECT AVG(NULLIF(COALESCE(current_year, previous_year), 0.00)) AS 'Average Budget' FROM budgets

Funktion	Syntax	Beispiel
PARSENAME	PARSENAME ('object_name', object_piece)	SELECT PARSE-NAME('pubs..authors', 1) AS 'Object Name' SELECT PARSE-NAME('pubs..authors', 2) AS 'Owner Name' SELECT PARSE-NAME('pubs..authors', 4) AS 'Server Name'

PERMISSIONS	PERMISSIONS ([objectid [, 'column']])	IF PERMISSIONS()&2=2
		  CREATE TABLE test_table (col1 INT)
		ELSE
		  PRINT 'ERROR: The current user cannot create a table.'

@@ROWCOUNT	@@ROWCOUNT	UPDATE authors SET au_lname = 'Jones' WHERE au_id = '999-888-7777' IF @@ROWCOUNT = 0 print 'Warning: No rows were updated'
ROWCOUNT_BIG	ROWCOUNT_BIG ( )	
SCOPE_IDENTITY	SCOPE_IDENTITY( )	SELECT SCOPE_IDENTITY() AS [SCOPE_IDENTITY] SELECT @@IDENTITY AS [@@IDENTITY]
SERVERPROPERTY	SERVERPROPERTY (propertyname)	SELECT CONVERT(char(20), SERVERPROPERTY ('servername'))
SESSIONPROPERTY	SESSIONPROPERTY (option)	SELECT SESSIONPROPERTY ('CONCAT_NULL_YIELDS_NULL')
SESSION_USER	SESSION_USER	DECLARE @session_usr char(30) SET @session_usr = SESSION_USER SELECT 'This session''s current user is: '+ @session_usr

Funktion	Syntax	Beispiel
STATS_DATE	STATS_DATE (table_id, index_id)	USE master SELECT 'Index Name' = i.name, 'Statistics Date' = STATS_DATE(i.id, i.indid) FROM sysobjects o, sysindexes i WHERE o.name = 'employee' AND o.id = i.id
SYSTEM_USER	SYSTEM_USER	DECLARE @sys_usr char(30) SET @sys_usr = SYSTEM_USER SELECT 'The current system user is: '+ @sys_usr
@@TRANCOUNT	@@TRANCOUNT	BEGIN TRANSACTION UPDATE authors SET au_lname = upper(au_lname) WHERE au_lname = 'White' IF @@ROWCOUNT = 2 COMMIT TRAN IF @@TRANCOUNT > 0 BEGIN PRINT 'A transaction needs to be rolled back' ROLLBACK TRAN END
USER_NAME	USER_NAME ([id])	SELECT USER_NAME(13)

B.2.9 Text- und Imagefunktionen

Funktion	Syntax	Beispiel
PATINDEX	PATINDEX ('%pattern%', expression)	SELECT PATINDEX ('%wonderful%', notes) FROM titles WHERE title_id = 'TC3218'
TEXTPTR	TEXTPTR (column)	DECLARE @ptrval var- binary(16) SELECT @ptrval = TEXTPTR(logo) FROM pub_info pr, publishers p WHERE p.pub_id = pr.pub_id AND p.pub_name = 'New Moon Books'

Funktion	Syntax	Beispiel
TEXTVALID	TEXTVALID ('table.column' , text_ptr)	SELECT pub_id, 'Valid (if 1) Text data' = TEXTVALID ('pub_info.logo', TEXTPTR(logo)) FROM pub_info ORDER BY pub_id

C Internetadressen

C.1 Normung

www.ansi.org
www.iso.ch
www.din.de

C.2 OLAP

www.olapreport.com

C.3 Sicherheit

www.bsi.de
www.microsoft.com/technet/security/tools.asp
www.pgp.com/products/freeware/default.asp
www.pgp.com/

C.4 Ergonomie

www.sozialnetz-hessen.de/ergo-online/software/inhalt.htm

C.5 XML

www.xmlsoftware.com

C.6 PostgreSQL

http://techdocs.postgresql.org/
www.postgresql.org

C.7 MS SQL Server

http://www.microsoft.com/sql/evaluation/trial/2000/default.asp

C.8 MS Access

www.microsoft.com/office/access/default.asp

C.9 Objektorientierte DBMS

www.odbmsfacts.com/articles/object-oriented_databases.html

C.10 Crystal Reports

www.crystaldecisions.com/

D HTML Kurzreferenz

D.1 Links und Inline-Images

	Link auf ein Dokument.
	Link auf ein Lesezeichen innerhalb desselben Dokumentes.
	Link auf ein Lesezeichen in einem anderen Dokument.
	Lesezeichen (Sprungziel).
<img src="URL">	Eingebundene Grafik.
<img src="URL" alt="***">	Eingebundene Grafik mit alternativem Text, der angezeigt wird, wenn die Grafik nicht mitgeladen wird.
<img src="URL" ismap>	aktive Grafik.
<img src="URL" align="top|middle|bottom">	Positionierung des Textes an der Grafik.
<imgsrc="URL"align="left|right|texttop| absmiddle|baseline|absbottom">	Steuerung des Textumflusses um die Grafik.
<img src="URL" border=?>	Rahmen um die Grafik in Pixel.
<img src="URL" lowsrc="URL">	geringe Auflösung für Proxy-Modus.
<img src="URL" width="?" height="?">	Größe der Grafik in Pixel.
<img src="URL" width="?%" height="?%">	Größe der Grafik in Prozent.
<img src="URL" hspace="?" vspace="?">	Abstand des Bildes vom nächsten Element in Pixel.

D.2 Client-Side Imagemaps

<map name="***"></map>	Client-Side-Imagemap-Rahmentag.
<area href="URL">	Ziel-URL des Area-Links.
<area alt="***">	Alternativtext für nichtgraphische Browser.

<area shape="rect|circle|poly" coords="left-x,top-y,right-x,bottom-y| center-x,center-y,radius| x1,y1,x2,y2,x3,y3,...">	Koordinatenwerte in Pixel (bei Prozentangaben werden die Werte relativ zu den "height"- und "width"-Werten des Bildes ausgelegt).

D.3 Formulare

<form action="URL" method="get|post"> </form>	Formulardefinition.
<input type="text|password|radio| checkbox|submit|reset">	Eingabefeldertypen.
<input name="***">	Feldname.
<input checked>	Vorgabe.
<input size="?">	Eingabefeldgröße (Zahl).
<input maxlength="?">	maximale Länge des Eingabefeldes.
<select></select>	Auswahl.
<select name="***"></select>	Listenname.
<select size="?"></select>	Anzahl der Optionen.
<select multiple></select>	multiple choice.
<option></option>	Optionen.
<option selected>?	Optionsvorgabe.
<textarea rows="?" cols="?"></textarea>	Texteingabefeld mit Größenvorgabe.
<textarea name="***"></textarea>	Feldname.

D.4 Tabelle

<table></table>	Tabellendefinition.
<table border="?"></table>	Tabellenlinienbreite.
<table cellpadding="?"></table>	Zellen(feld)höhe in Pixel.
<table cellspacing="?"></table>	Zellen(feld)breite.
<table width="?"></table>	Tabellenbreite in Pixel.
<table width="?%"></table>	Tabellenbreite in Prozent.
<table bgcolor="#$$$$$$"></table>	Hintergrundfarbe für alle Tabellenfelder.
<tr></tr>	Tabellenreihe.
<tr align="left|right|center">	vert. Positionierung des Inhalts in der Reihe.
<tr valign="top|middle|bottom">	hor. Positionierung des Inhalts in der Reihe.
<tr bgcolor="#$$$$$$"></tr>	Hintergrundfarbe für die Tabellenreihe.

<td></td>	Tabellenfeld (-zelle).
<td align="left|right|center">	vert. Positionierung des Inhalts in dem Feld.
<td valign="top|middle|bottom">	hor. Positionierung des Inhalts in dem Feld.
<td nowrap></td>	kein Zeilenumbruch.
<td colspan="?"></td>	Spaltenanzahl, über die das Feld reichen soll.
<td rowspan="?"></td>	Reihenanzahl, über die das Feld reichen soll.
<td width="?"></td>	Feldbreite in Pixel.
<td width="?%"></td>	Feldbreite in Prozent.
<td bgcolor="#$$$$$$"></td>	Hintergrundfarbe für das Tabellenfeld.
<th></th>	Überschriftsfeld (Header-Feld).
<th align="left|right|center">	vert. Positionierung des Inhalts.
<th valign="top|middle|bottom">	hor. Positionierung des Inhalts.
<th nowrap></th>	kein Zeilenumbruch.
<th colspan="?"></th>	Spaltenanzahl, über die das Feld reichen soll.
<th rowspan="?"></th>	Reihenanzahl, über die das Feld reichen soll.
<th width="?"></th>	Feldbreite in Pixel.
<th width="?%"></th>	Feldbreite in Prozent.
<th bgcolor="#$$$$$$"></th>	Hintergrundfarbe für das Tabellenüberschriftsfeld.
<caption></caption>	Tabellenbezeichnung (Über-/Unterschrift).
<caption align="top|bottom">	Positionierung der Tabellenbezeichnung.
<caption center></caption>	Überschrift mittig.

D.5 Sonderzeichen

>		>	größer als
<		<	kleiner als
&		&	Kaufmanns-Und
"		"	Anführungszeichen
ä	ä	Ä	Ä
ö	ö	Ö	Ö
ü	ü	Ü	Ü
ß	ß		

D.6 Sonstiges

<!-- *** -->	Kommentar (wird vom Browser nicht angezeigt)
<embed src="URL" width="?" height="?" autostart="true|false" loop="true|false">	Plugin-Schnittstelle (z.B. für Soundfiles)

E Literaturangaben

Vossen G. (1994). Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme /2. Auflage: Addison Wesley; ISBN 3-89319-566-1

Vossen G. (1999). Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme /3. Auflage: Oldenbourg; ISBN 3-486-24544-9

Wessel I. (1998). Richtlinien zur Gestaltung ergonomischer Windowsapplikationen: Hanser; ISBN 3-446-19389-8

Doberenz W., Kowalski T. (1999). Datenbankprogrammierung mit Visual Basic 6: Microsoft Press; ISBN 3-86063-485-2

Elzer P. (1994). Management von Softwareprojekten: eine Einführung für Studenten und Praktiker: Vieweg; ISBN 3-528-05400-X

Trauboth H. (1996). Software-Qualitätssicherung: konstruktive und analytische Maßnahmen: Oldenbourg; ISBN 3-486-23412-9

Monadjemi P. (2000). Ofiice 2000 Developer Edition: Markt und Technik; ISBN 3-8272-5514-7

Clausen N. (1998). OLAP – Multidimensionale Datenbanken: Addison Wesley; ISBN 3-8273-1402-X

Gutierrez D. (2000). Web-Datenbanken – für Windows-Plattformen: Markt und Technik; ISBN 3-8272-5774-3

Dicken H. (2000). Datenbanken unter Linux: MITP; ISBN 3-8266-0555-1

Kofler M. (1998). Visual Basic 6: Addison Wesley; ISBN 3-8273-1428-3

Anahory, Sam (1997). Data Warehouse: Addison Wesley; ISBN 3-8273-1288-4

F Installation von PostgreSQL

In diesem Abschnitt finden Sie eine Kurzanleitung für die Installation von PostgreSQL unter Windows NT oder Windows 2000.

F.1 Installation des Cygwin-Paketes

Das Cygwin-Paket bietet eine Unix-ähnliche Programmierschnittstelle für Windows-Umgebungen.

F.1.1 Datenquelle

Sie finden die Installationsdateien auf der Begleit-CD im Ordner /CYGWIN oder alternativ im Internet unter der Adresse *http://sources.redhat.com/cygwin/*

F.1.2 Vorgehensweise

1. Kopieren Sie den Ordner /LATEST in ein temporäres Verzeichnis.
2. Starten Sie das Setup und geben Sie den Ordner /LATEST als Quellverzeichnis an.
3. Folgen Sie den weiteren Anweisungen.
4. Für Minimalinstallation sind zumindest die Pakete CYGWIN, FILEUTILS und W32API zu installieren.

F.2 Installation des Cygipc-Paketes

Cygipc ist ein Erweiterungspaket für Cygwin und stellt weitere Funktionen für Speicherverwaltung, Zeichenträger und Meldungen bereit.

F.2.1 Datenquelle

Sie finden die Installationsdateien auf der Begleit-CD im Ordner /CYGIPC oder, wenn Sie die aktuellste Version aus dem Internet laden wollen, unter der Adresse: *http://www.neuro.gatech.edu/users/cwilson/cygutils/v1.1/cygipc/*

F.2.2 Vorgehensweise

1. Entpacken Sie das Archiv CYGIPC-1.09-2.TAR unter /CYGWIN

```
$ tar -xvsf <path/cygipc-1.09-2.tar>
```

2. Starten Sie den Dienst IPC-DAEMON

```
$ ipc-daemon -install-as-service
$ net start ipc-daemon
```

F.3 Installation von PostgreSQL

Schließlich können Sie mit der Installation von PostgreSQL beginnen. Führen Sie dazu die genannten Schritte aus oder verwenden Sie die Dokumentation auf der Begleit-CD.

F.3.1 Datenquelle

Die für die Installation notwendigen Dateien der Version 7.1.3 finden Sie auf der Begleit-CD im Ordner /POSTGRESQL oder alternativ im Internet unter der Adresse:

http://www.postgresql.org

F.3.2 Vorgehensweise

1. Kopieren und entpacken Sie das Archiv im Verzeichnis /USR/SRC:

```
$ tar -xvsf  /usr/src / postgresql-7.1.3.tar
```

2. Legen Sie zusätzliche Ordner an:

```
$ mkdir -p /usr/local/pgsql/{bin,include,lib,data}
```

3. Wechseln Sie in das PostgreSQL-Verzeichnis:

```
$ cd /usr/src/postgresql-7.1.3/src
```

4. Führen Sie das Konfigurationsskript aus:

```
$./configure
```

5. Starten Sie die Installationsroutine:

```
$ make
$ make install
```

6. Erzeugen Sie die Datei .BASHRC mit Hilfe eines Texteditors und dem folgenden Inhalt:

```
PATH=$PATH:/usr/local/pgsql/bin:/usr/local/bin
PGDATA=/usr/local/pgsql/data
PGLIB=/usr/local/pgsql/lib

LD_LIBRARY_PATH=/usr/local/pgsql/lib:/usr/local/lib
```

7. Initialisieren Sie die Datenbank:

```
$ initdb
```

8. Ändern Sie die IP-Adressen in der Konfigurationsdatei PG_HBA.CONF im Verzeichnis /USR/LOCAL/PGSQL/DATA:

   `Host all 163.17.11.109 255.255.255.0` (Beispiel)

9. Starten Sie PostgreSQL:

```
$ psql -h host_name template1
```

Für die Installation von PostgreSQL, welche zugegebenermaßen nicht ganz einfach ist, steht Ihnen die Hilfe im Ordner /POSTGREHELP zur Verfügung. Öffnen sie die Datei ADMIN.HTM in Ihrem Browser und navigieren Sie durch den Administrationsteil von PostgreSQL. Außerdem finden Sie einige Berichte über mögliche Probleme bei der Installation im Internet.

Index