Einführung in Relationale Datenbanken und SQL

Auszug aus:
Thorsten Strobel: *Entwicklung von Datenbankkomponenten für eine relationale Sybase-SQL-Datenbank*. Diplomarbeit Nr. 1664 am Institut für Automatisierungs- und Softwaretechnik, Universität Stuttgart, 1998.

Inhaltsverzeichnis

1 Grundlagen

In [MAR94] wird der Begriff „Datenbank„ folgendermaßen beschrieben:

„Eine Datenbank ist ein integriertes Ganzes von Datensammlungen, aufgezeichnet in einem nach verschiedenen Gesichtspunkten direkt zugänglichen Speicher, für Informationsbeschaffung in großem Umfang bestimmt, verwaltet durch ein separates Programmsystem.„

Diese zusammenhängende Menge von Daten bezeichnet man als *Datenbasis*. Das Programmsystem, welches die Datenbasis verwaltet, wird *Datenbankmanagementsystem* (DBMS) genannt. Es führt sämtliche Lade- und Speichervorgänge auf der Datenbasis durch. Es besteht also keine Möglichkeit des direkten Zugriffs auf die Datenbasis. Durch dieses Programmsystem erfolgt eine Trennung zwischen den Anwenderprogrammen und den Daten. Dieser Sachverhalt ist in Abbildung 1-1 dargestellt .

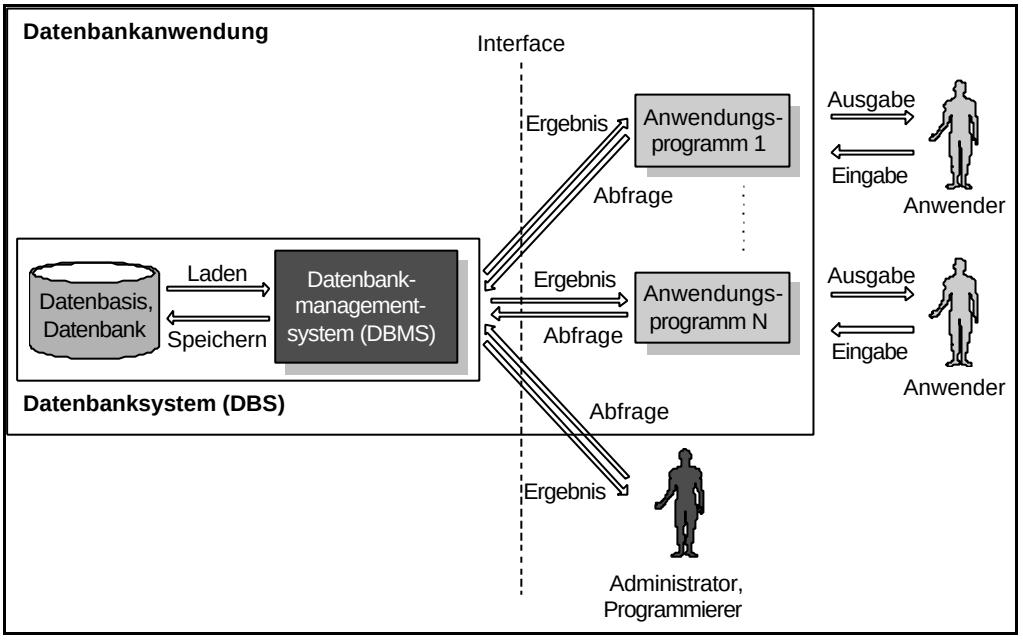


Abbildung 1-1 Schema eines Datenbanksystems

Datenbasis und Datenbankmanagementsystem zusammen bilden das *Datenbanksystem* (DBS).

Der Zugriff eines Anwenders auf die Daten über das DBMS erfolgt in der Regel durch Anwendungsprogramme. Diese bieten dem Anwender durch eine entsprechende (meist graphische) Benutzungsoberfläche vielfältige Möglichkeiten zur Interaktion mit dem Datenbanksystem. Die Eingaben des Anwenders werden dabei vom Anwendungsprogramm in Abfragen an das DBMS umgesetzt. Dieses führt die Abfragen aus und liefert das resultierende Ergebnis an das Anwendungsprogramm zurück, welches daraus eine Ausgabe an den Anwender erstellt. Anwendungsprogramm und Datenbanksystem sind dabei über eine definierte Schnittstelle (interface) verbunden. Anwendungsprogramm(e) und Datenbanksystem zusammen werden als *Datenbankanwendung* bezeichnet.

Datenbankadministratoren und Anwendungsprogrammierer können über eine geeignete Schnittstelle auch direkt auf das DBMS zugreifen, um zum Beispiel Änderungen an der Datenbasis oder den Zugriffsrechten vorzunehmen.
Da sich je nach Zusammenhang hinter dem Begriff „Datenbank„ die Datenbasis, das Datenbanksystem oder die Datenbankanwendung verbirgt, wird im Weiteren auf die Verwendung dieses Begriffes verzichtet. Statt dessen werden die exakten Begriffe wie oben stehend verwendet.

Es gibt verschiedene Konzepte für Datenbanksysteme. Am weitesten verbreitet sind relationale Datenbanksysteme. Daneben existieren noch hierarchische und Netzwerk-Datenbanksysteme. Hierarchische Datenbanksysteme ordnen die Daten in Baumstrukturen an. Netzwerkdatenbanksysteme erlauben zusätzlich zur hierarchischen Anordnung der Daten auch beliebige Querverbindungen zwischen den Daten. Um bestimmte Daten in einem hierarchischen Datenbanksystem oder in einem Netzwerkdatenbanksystem zu finden, muß der Benutzer durch diese Strukturen navigieren. Das Konzept der relationalen Datenbanksysteme wird in Kapitel 4.4 ausführlich erläutert.

Um die Anforderungen der objektorientierten Programmierung und Entwicklung zu berücksichtigen, werden verstärkt objektrelationale Datenbanksysteme und Objektdatenbanksysteme eingesetzt. Objektrelationale Datenbanksysteme verwenden die Technik der relationalen Datenbanksysteme und erweitern diese um die Verarbeitung von Objekten. Objektdatenbanksysteme gehen dagegen eigene, meist von Hersteller zu Hersteller verschiedene Wege bei der Verwaltung von Objekten.

1975 wurde von der ANSI/X3/SPARC-Gruppe (American National Standards Institute/ Computer and Information Processing/Standards Planning and Requirements Committee) ein Report erstellt, wie ein Datenbanksystem sein soll. Dabei ist die beschriebene Architektur (siehe Abbildung 1-2) unabhängig von dem zugrundeliegenden Konzept (relational, hierarisch, Netzwerk, ...) des Datenbanksystems.

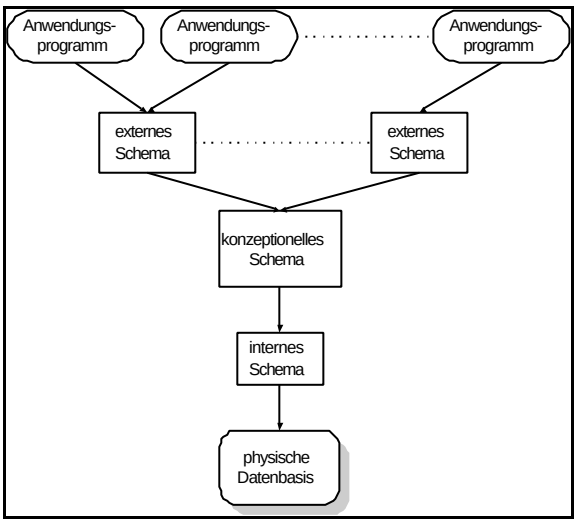


Abbildung 1-2 ANSI/SPARC-Architektur

Das konzeptionelle Schema[1] beschreibt die gesamte Datenstruktur einer Datenbankanwendung. Dabei ist unerheblich, auf welche Weise diese Daten abgelegt werden. Das konzeptionelle Schema dient dem

[1] Das konzeptionelle Schema darf nicht dem konzeptionellen oder logischen Entwurf bei der Entwicklung der Datenbasis verwechselt werden. Das konzeptionelle Schema bei einem relationalen DBS ist die mit SQL implementierte Gesamtstruktur der Datenbasis inklusive der Konsistenzbedingungen wie Fremdschlüssel und Trigger. Die externen Schemata werden bei relationalen DBMS mit Hilfe von Views realisiert (näheres in der Beschreibung von SQL, Kapitel 5). Das interne Schema bleibt dem Anwendungsprogrammierer in der Regel verborgen.

DBMS als logische Grundlage für seine Operationen auf der Datenbank. Neben der Beschreibung der Datenstruktur gehören vor allem auch Konsistenzbedingungen zum konzeptionellen Schema.

Im internen Schema sind die Einzelheiten der Implementierung zu beschreiben, z.B. Aufteilung der Datenbank auf verschiedene Dateien. Das interne Schema befindet sich noch über der physischen Datenbasis, da von einem linearen Adressraum ausgegangen wird und gerätespezifische Parameter wie Blöcke, Zylinder und Spuren außer acht gelassen werden.

In den externen Schemata werden Teilbereiche der Datenbank für verschiedene Anwendungsprogramme definiert. Hier werden auch die Zugriffsrechte der einzelnen Anwendungen bzw. Benutzer festgelegt.

2 Entwicklung einer Datenbankanwendung

Prinzipiell läßt sich die Entwicklung einer Datenbankanwendung in die üblichen Phasen der Softwareentwicklung einteilen ([LUD97]). Diese Phasen und die Entwicklungsschritte sind in Abbildung 2-1 dargestellt.

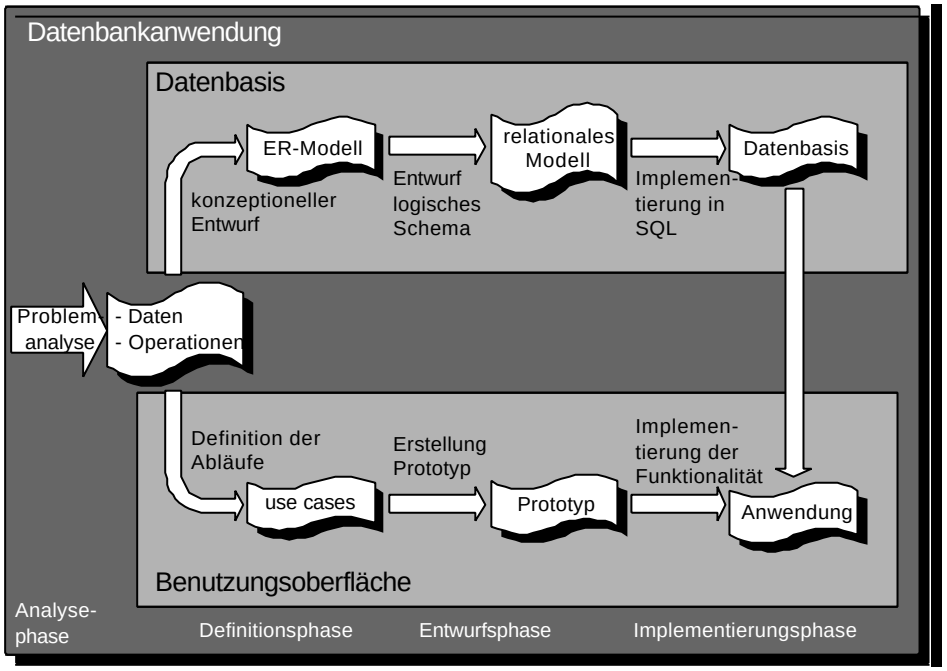


Abbildung 2-1 Entwicklungsphasen und -schritte

Die *Problemanalyse* in der Analysephase dient der Sammlung der Anforderungen und liefert die zu speichernden Daten und die darauf auszuführenden Operationen aus der Sicht einzelner Benutzergruppen.

Danach teilt sich die Entwicklung in zwei fast unabhängige Schritte auf. Im ersten Teil findet die Entwicklung der *Datenbasis* statt. Die zu speichernden Daten werden im *konzeptionellen Entwurf* (siehe Kapitel 3) aus der Sicht der einzelnen Benutzertypen (z.B. Finanzverwalter, Beschaffer, ...) modelliert. Dazu wird i.d.R. die Entity-Relationship-Modellierung (siehe Kapitel 3.2) verwendet. Diese Teilmodelle werden dann zu einem Gesamtmodell zusammengefügt ([RLE97]), wobei auftretende Strukturkonflikte gelöst werden müssen. Dieses konzeptionelle Modell ist noch völlig unabhängig von der Art des verwendeten Datenbanksystems.

Im *Entwurf des logischen Schemas* (siehe Kapitel 4) wird das konzeptionelle Modell in ein implementierungsabhängiges Modell transformiert. Diese Beschreibung der Datenstrukturen auf logischer Ebene nennt man *Datenmodell*. Da ein relationales Datenbanksystem zum Einsatz kam, wurde an dieser Stelle das relationale Datenmodell verwendet. Um Redundanzen und Anomalien bei Speicheroperationen zu vermeiden, werden die Tabellen des relationalen Modells normalisiert.

Anschließend wird das relationale Datenmodell mit Hilfe von SQL im Datenbanksystem angelegt.

Der zweite Teil der Entwicklung beinhaltet die graphische Benutzungsoberfläche. Die relevanten Abläufe der zu entwickelnden Anwendung, welche sich aus der Problemanalyse ergeben haben, werden z.B. mit Hilfe von Use Cases beschrieben.

Um die späteren Benutzer bereits in einer frühen Phase in die Entwicklung mit einbeziehen zu können, wird ein Prototyp der Benutzungsoberfläche erstellt. Dieser Prototyp ermöglicht dem Benutzer einen Einblick in die Bedienung der Anwendung.

Im letzten Entwicklungsschritt wird die volle Funktionalität der Anwendung mit Hilfe einer geeigneten Entwicklungsumgebung implementiert. Hierbei wird auf die bereits implementierte Datenbasis aufgesetzt.

3 Konzeptioneller Entwurf / ER-Modellierung

3.1 Vorgehen

Bei der Problemanalyse werden die Anforderungen und Wünsche der einzelnen Benutzertypen wie Finanzverwalter, Hiwiverwalter, usw. ermittelt. Daraus ergeben sich die zu speichernden Daten aus der Sicht dieser Benutzer. Die Struktur dieser Daten wird mit Hilfe der Entity-Relationship-Modellierung dargestellt, indem Teilmodelle für die einzelnen Benutzersichten angelegt werden. Dabei kann es natürlich auch zu Überlappungen kommen, wenn bestimmte Daten für unterschiedliche Benutzertypen relevant sind. Diese gemeinsam genutzten Daten dienen bei der anschließenden Vereinigung der Teilmodelle besonderer Beachtung, da es hier zu sogenannten Strukturkonflikten kommen kann.

Ein solcher Strukturkonflikt kann zum Beispiel entstehen, wenn derselbe Teil der zu modellierenden Daten durch unterschiedliche ER-Konstrukte beschrieben wird. Sind diese verschiedenen Konstrukte kompatibel zueinander, kann der Strukturkonflikt durch Vereinheitlichung der Darstellungen gelöst werden. Sind die Konstrukte dagegen inkompatibel, weil zum Beispiel Beziehungen unterschiedliche Kardinalitäten besitzen, so muß entweder eine Darstellung zugunsten der anderen aufgegeben werden oder müssen Integritätsbedingungen die Konsistenz sicherstellen.

Auch Namenkonflikte können auftreten, wenn dieselben Daten unterschiedlich benannt werden. Diese Synonyme lassen sich durch Umbenennen entfernen. Existieren dagegen unterschiedliche Konstrukte mit dem selben Namen, spricht man von Homonymen. Diese werden dadurch aufgelöst, indem man jedes Konstrukt mit unterschiedlichen Namen anlegt.

3.2 Entity-Relationship-Modellierung

Zur Modellierung der in der Datenbank zu speichernden Daten wird in der Definitionsphase bei relationalen Datenbanksystemen in der Regel die Entity-Relationship(ER)-Modellierung eingesetzt, welche von P. Chen [CHE91] 1976 entwickelt wurde. Später wurde die ER-Modellierung durch zusätzliche Konzepte zur semantischen Datenmodellierung erweitert. Außerdem existieren verschiedene Formen der Darstellung von ER-Modellen. Deshalb soll vor allem die verwendete Form der ER-Modellierung und nicht die historische Entwicklung dieser Modellierung erläutert werden.

Ein ER-Diagramm besteht aus Entitäten (*Entities*) und Beziehungen (*Relationships*).

Eine *Entität* ist ein individuelles, unterscheidbares und identifizierbares Exemplar von Dingen, Personen oder Begriffen aus der realen oder der Vorstellungswelt ([MDA97]). Hinsichtlich ihrer beschreibenden Eigenschaften homogene Entitäten können zu einem *Entitätstyp* zusammengefaßt werden, der mit einem eindeutigen Oberbegriff (ein Subjektiv im Singular) benannt wird.

Zwei oder mehrere Entitäten können wechselseitig miteinander durch *Beziehungen* verbunden sein. Eine Beziehung ist ein unselbständiges Objekt, es existiert nur durch die konstituierenden Objekte. Die Entitäten in einer Beziehung spielen jeweils eine *Rolle*. Beziehungen, welche dieselben Rollen und Eigenschaften haben, werden zu eindeutig benannten (Verb im Singular) *Beziehungstypen* (*Assoziationen*), zusammengefaßt. Da durch die Rollen der Entitäten ein Beziehungstyp meist exakter beschrieben wird als durch die Benennung, kann auf diese auch verzichtet werden.

Die möglichen Typen (*Kardinalitäten*) von Assoziationen sind:

- 1:1-Assoziation
  Zwei Entitätstypen A und B stehen zueinander in einer 1:1-Assoziation, wenn jede Entität aus A mit genau einer Entität aus B verbunden ist und umgekehrt.
- 1:M-Assoziation
  Zwei Entitätstypen A und B stehen zueinander in einer 1:M-Assoziation, wenn es zu jeder Entität aus A eine oder mehrere Entitäten in B gibt, zu jeder Entität aus B aber genau eine Entität in A existiert.
- N:M-Assoziation
  Zwei Entitätstypen A und B stehen zueinander in einer N:M-Assoziation, wenn zu jede Entität aus A eine oder mehrere Entitäten in B vorhanden sind und umgekehrt.

N bzw. M bedeuten dabei beliebige Obergrenzen, wobei N und M auch identisch sein können. Außerdem können auch Entität A und Entität B identisch sein (*rekursive Beziehung*).

Um auch optionale Beziehungen modellieren zu können, wurden *konditionelle Assoziationen* eingeführt:

- 1:C-Assoziation
  Zwei Entitätstypen A und B stehen zueinander in einer 1:C-Assoziation, wenn jede Entität aus A mit einer Entität aus B oder mit keiner Entität verbunden ist. Zu jeder Entität aus B existiert jedoch genau eine Entität aus A.
- 1:MC-Assoziation
  Zwei Entitätstypen A und B stehen zueinander in einer 1:MC-Assoziation, wenn jede Entität aus A mit einer oder mehreren Entitäten aus B oder mit keiner Entität verbunden ist. Zu jeder Entität aus B existiert jedoch genau eine Entität aus A.
- N:MC-Assoziation

Zwei Entitätstypen A und B stehen zueinander in einer N:MC-Assoziation, wenn jede Entität aus A mit einer oder mehreren Entitäent aus B oder mit keiner Entität verbunden ist. Zu jeder Entität aus B existiert eine oder mehrere Entitäten aus A.

Es sind auch Kombinationen aus den einzelnen Arten von Beziehungen möglich, wie z.B. C:C-Assoziation.

Entitäten und Beziehungen besitzen Eigenschaften, die durch Paare von *Attribut* und Wert ausgedrückt werden können. Zum Beispiel könnte das Attribut „Alter" einer Entität „Person X" den Wert „24" besitzen. Der *Wertebereich* (*Domäne*) gibt die Menge aller möglichen Werte für ein Attribut an. In manchen Fällen kann ein Attribut mehr als einen Wert für eine gegebene Entität haben, wie z.B. das Attribut „Telefonnummer" der Entität „Person X".

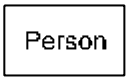
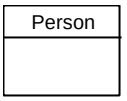
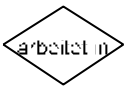
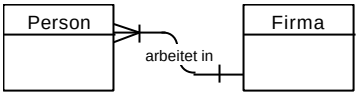
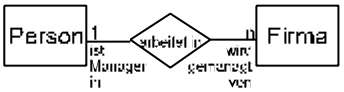
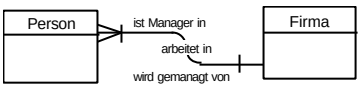
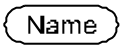
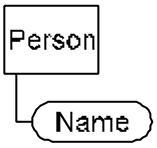
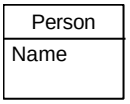
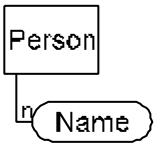
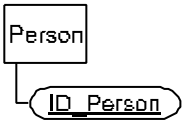
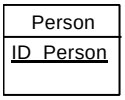
Die Identifizierung von Entitäten geschieht durch den Gebrauch von Attribut-Werte-Paaren. Die Kombinationen von Attributen, welche eine Entität identifizieren können, werden *Schlüssel* (*Schlüsselkandidaten*) genannt. Dabei wird von einem Schlüssel Minimalität verlangt, was bedeutet, daß er nur soviel Attribute enthält, wie zur eindeutigen Identifizierung notwendig sind. Oft ist es allerdings nötig, daß künstliche Attribute zur Identifizierung eingesetzt werden, wenn die vorhandenen Attribut-Werte-Paare nicht eindeutig sind.

Bei sogenannten schwachen Entitätstypen ist die Existenz von der Existenz eines anderen Existenztypen abhängig. Dies wird *Existenzabhängigkeit* genannt. Bei der *Schlüsselabhängigkeit* geschieht die Identifizierung einer Entität nicht durch die eigenen Attribute, sondern durch ihre Beziehungen mit anderen Entitäten. So ist z.B. eine Straße nur innerhalb einer Stadt eindeutig.

Um auszudrücken, daß eine Entität zu mehreren Entitätstypen gehören kann, wurde in einer Erweiterung des ER-Modells die *Generalisierung* bzw. *Spezialisierung* (oft auch als „Vererbung„ bezeichnet) eingeführt. Dadurch können mehrere Entitätstypen zu einem übergeordneten Entitätstyp (Superentitätstyp) zusammengefasst werden, welcher die redundanten, d.h. in allen untergeordneten (spezialisierten) Entitätstypen (Subentitätstyp) vorhandenen Attribute enthalten kann. Eine Entität, die durch den Subentitätstyp beschrieben wird, stellt dieselbe Entität dar, die durch den Superentitätstyp beschrieben wird. Sie kann lediglich zusätzliche Attribute enthalten. Eine solche Beziehung wird aus der Sicht der Subentität auch als *ist-eine-Beziehung* bezeichnet. Es existieren vier mögliche Arten von Generalisierung:

- total/disjunkt: jede Superentität wird durch eine Subentität ausgedrückt (z.B. Superentität Mitarbeiter, Subentitäten Mann, Frau. Jeder Mitarbeiter ist entweder Mann oder Frau)
- partiell/disjunkt: Superentitäten werden entweder durch keine oder genau eine Subentität ausgedrückt (z.B. Superentität Kind, Subentitäten Bub, Mädchen. Jedes Kind ist entweder Bub oder Mädchen oder keines von beiden (Embryo))
- total/nicht disjunkt: eine Superentität kann durch eine oder mehrere Superentitäten ausgedrückt werden (z.B. Superentität Partner, Subentitäten Kunde, Lieferant. Jeder Partner ist Kunde, Lieferant oder beides)
- partiell/nicht disjunkt: eine Superentität kann durch keine, eine oder mehrere Superentitäten ausgedrückt werden (z.B. Superentität Person, Subentitäten Arzt, Patient. Jede Person ist entweder Arzt, Patient , beides oder keines von beiden (Sekretär))

Es existieren viele verschiedene Notationen für ER-Modelle. In Tabelle 3-1 ist sowohl die in Lehrbüchern verwendete Notation also auch die von den gängigen Modellierungswerkzeugen unterstütze Krähenfußnotation dargestellt.

Strukturelement	Lehrbuchnotation	Krähenfußnotation
Entitätstyp	 Rechteck mit Bezeichnung	 Rechteck mit Bezeichnung als Titel
Beziehungstyp	 Raute mit Bezeichnung  Verbindung mit Entitäten durch Linien	 beschriftete Linie, verbindet Entitäten 
Rolle einer Entität	 Beschriftung der Verbindungslinie	 Beschriftung der Verbindungslinie
Attribut	 Oval mit Bezeichnung, 	 im Rechteck der Entität
mehrwertiges Attribut	 Beschriftung der Verbindungslinie	Darstellung nicht möglich
Primärschlüssel	 Attribut unterstrichen	 Attribut unterstrichen

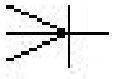
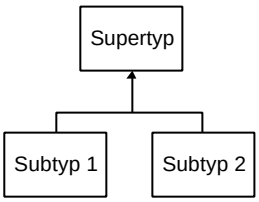
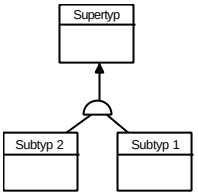
Kardinalität der Assoziationen	1 Beschriftung der Verbindungslinie	 spezielle Endungen der Verbindungslinien
Kardinalität 1	1	
Kardinalität M	M	
Kardinalität C	C	
Kardinalität MC	MC	
Generalisierung		

Tabelle 3-1 ER-Notation

An einem kleinen Beispiel (Abbildung 3-1) soll nun der Aufbau eines ER-Modells erklärt werden.

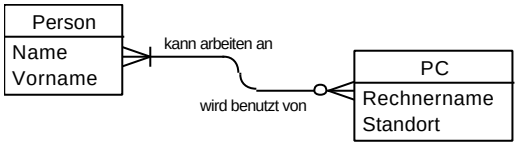


Abbildung 3-1 Beispiel ER-Modell

In diesem Modell existieren Entitäten vom Typ Person mit den Merkmalen (Attributen) Name und Vorname. Außerdem sind Entitäten vom Typ PC mit dem Merkmal Rechnername und Standort vorhanden. Auf die Darstellung von Schlüsseln soll an dieser Stelle der Einfachheit halber verzichtet werden. Zwischen diesen beiden Entitätstypen besteht eine N:MC-Assoziation. Eine Person kann an einem, mehreren oder auch an gar keinem PC arbeiten. Jeder PC wird von mindestens einer Person benutzt.

4 Entwurf des logischen Schemas / Relationales Modell

4.1 Das relationale Modell

Das relationale Modell (auch als *Relationenmodell* bezeichnet) für Datenbanken wurde 1970 von E. F. Codd vorgestellt und 1990 erweitert. Es ist das am besten untersuchte Datenmodell. Da es auf der *Relationenalgebra* und dem *Relationenkalkül* beruht, ist es exakt formulierbar. Das relationale

Datenmodell verwendet Begriffe, deren Namen auch in anderen Modellierungsmethoden (wie z.B. der Entity-Realtionship-Modellierung) vorkommen, dort aber teilweise eine unterschiedliche Bedeung besitzen. Um Mißverständnissen vorzubeugen, muß diesen Begriffen besondere Aufmerksamkeit gewidmet werden.

Strukturbezogene Informationen werden in diesem Datenmodell ausschließlich durch den Inhalt des einzigen Strukturelements, der *Relation*, ausgedrückt. Eine Relation ist eine Menge gleichartiger Tupel.

Da die Gesamtheit der Tupel dem Betrachter in Tabellenform dargeboten wird, bezeichnet man Relationen oft auch als *Tabellen*[2]. Ein Tupel entspricht bei der tabellenartigen Darstellung einer Tabellenzeile und wird auch als Datensatz bezeichnet. Eine Tabellenspalte entspricht einem *Attribut* (anlog dem Entity-Relationship-Modell).

Ein *Tupel* ist eine Liste mit einer genau festgelegten Anzahl von Werten. Ein Tupel wird auch *Datensatz* genannt. Ein *Wert* stellt eine Ausprägung einer einzelnen Eigenschaft (Attribut) einer Entität dar.

In einer Relation bezeichnet die *Kardinalität* die Anzahl der Zeilen. Die Anzahl der Spalten wird durch den *Rang* (degree) oder die *Stellenzahl* (arity) ausgedrückt.

Die Identifizierung eines Tupels in einer Relation geschieht wie bei der ER-Modellierung durch *Schlüsselkandidaten*. Ein Schlüsselkandidat ist auch hier ein Attribut (unter Umständen auch mehrere, man spricht dann von einem zusammengesetzten Schlüsselkandidat), mit dessen Wert ein Tupel eindeutig identifiziert werden kann. Von einem Schlüsselkandidat wird verlangt, daß er nur soviele Attribute enthält, wie zur eindeutigen Identifizierung notwendig sind.

In einem Datenbanksystem muß ein Schlüsselkandidat zur Identifitierung festgelegt werden. Dieser wird als *Primärschlüssel* bezeichnet. Ist durch die vorhandenen Attribute keine eindeutige Identifizierung eines Tupels innerhalb einer Relation möglich, wird ein künstliches Attribut als Primärschlüssel eingeführt. Ein weiterer Grund für den Einsatz künstlicher Attribute ist die Forderung, daß der Primärschlüssel nicht geändert werden darf.

Während Domänen in der ER-Modellierung meistens vernachlässigt werden, sind sie im relationalen Modell unbedingt notwendig. Als *Domäne* bezeichnet man auch hier einen Wertebereich für ein Attribut, sozusagen einen Pool zulässiger Werte. Domänen sind deshalb so wichtig, weil sie Vergleiche von Attributen beschränken bzw. erst sinnvoll machen. So sollten nur Attribute verglichen werden können, die auf derselben (oder einer kompatiblen) Domäne definiert sind. Vergleicht man zum Beispiel, ein Attribut mit einem Zahlenbereich als Domäne mit einem Attribut, welches alphanumerische Zeichen als Wertebereich besitzt, macht dies keinen Sinn und kann sogar zu Fehlern führen. Als Domänen werden meistens die Datentypen benützt, welche vom verwendeten Datenbanksystem zur Verfügung gestellt werden.

[2] Genau genommen sind Relationen und Tabellen nicht dasselbe (siehe [DAT95]). Eine Relation ist ein abstraktes Objekt, eine Tabelle ist die konkrete Darstellung davon. Eine Relation unterscheidet sich von einer Tabelle dadurch, daß keine Ordnung der Tupel von oben nach unten und keine Ordnung der Attribute von links nach rechts wie in einer Tabelle vorhanden ist. Außerdem dürfen Tupel in einer Relation nicht mehrfach vorkommen, was in einer Tabelle ohne Primärschlüssel erlaubt ist. Weiterhin sind die Attribute in einer Relation alle atomar, was nur für Tabellen in der ersten Normalform gilt.

Ein Relationstyp zusammen mit dem Wertebereich der Attribute und den Abhängigkeiten der Attribute untereinander wird als Relationenschema bezeichnet.

Da nicht unbedingt jeder Benutzer auf alle Daten einer Relation zugreifen können sollte, andererseits aber manche Benutzer für ihre Anwendung Daten mehrerer Relationen benötigen, besteht im relationalen Datenmodell die Möglichkeit, Views zu definieren. Ein *View* ist eine besondere Form einer Relation, deren Inhalt durch Abfragen aus anderen Relationen gewonnen wird. Durch Views (benannte, abgeleitete, virtuelle Relationen [DAT95]) werden nur bestimmte Teile einer Datenbasis gezeigt, ohne daß diese nochmals gespeichert werden müssen, was überflüssige Redundanz (und damit auch mögliche Inkonsistenz) vermeidet. Der Zugriff auf diese virtuellen Relationen erfolgt weitgehend analog dem auf reale Relationen, allerdings können nicht in allen Views Änderungen und Einfügeoperationen ausgeführt werden. Vor allem bei Views, die sich aus mehreren Relationen zusammensetzen, ist die Änderung der Daten problematisch. Außerdem ist es durch die Benutzung von Views möglich, Anwenderprogramme, die sich nur auf Views beziehen, unverändert zu lassen, auch wenn sich die reale Struktur der Relationen ändert. Es muß dann nur eine Anpassung der Views erfolgen. Damit dienen Views hauptsächlich zur Realisierung der in Kapitel 0 beschriebenen externen Schemata.

Es existieren verschiedene Formen der Darstellung einer Relation bzw. Tabelle in einem relationalen Modell. Für eine knappe Darstellung wird oft die Textform gewählt. Dabei folgen dem Name der Relation die Attribute, zusammengefaßt als Tupel. Das Attribut (evtl. mehrere) des Primärschlüssels wird unterstrichen. Abbildung 4-1 zeigt ein Beispiel für die Textform einer Relation.

Person (ID_Person, Name, Vorname)

Abbildung 4-1 Textform einer Relation

Um die (allerdings nur durch den Inhalte ausgedrückte) Beziehung zwischen Relationen aufzeigen zu können, wird die graphische Form der Darstellung benützt. Abbildung 4-2 zeigt ein Beispiel für die graphische Darstellung einer Relation, wobei auch hier das Attribut des Primärschlüssel unterstrichen ist. Dabei enthält die Kopfzeile den Namen der Relation, die Zeilen des Rumpfes die einzelnen Attribute. Bei Bedarf kann hinter dem Namen des Attributs noch der Wertebereich bzw. der Datentyp angezeigt werden.

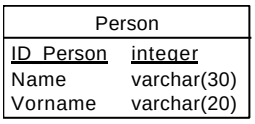


Abbildung 4-2 Graphische Form einer Relation

Um die Beziehung (keine echte Verbindung) zwischen den Relationen des relationalen Modells darzustellen, werden Pfeile benützt (siehe Abbildung 4-3). Die Pfeilspitze zeigt dabei in Richtung der Detailrelation (siehe Kapitel 4.2). Außerdem wird durch die Pfeilbeschriftung angezeigt, mit welchen Attributen eine Beziehung realisiert werden kann.

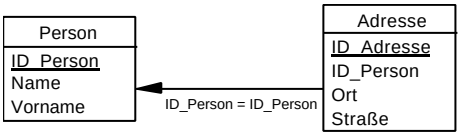


Abbildung 4-3 Beziehung zwischen Relationen

Auf Relationen können verschiedene Operationen ausgeführt werden ([BAL96]). Durch diese Operationen entstehen stets wieder Relationen. Wie solche Operationen in einer

Datenmanipulationssprache (z.B. SQL) des Datenbankmanagementsystems realisiert werden können, wird in Kapitel 5 beschrieben.

Eine *Selektion* wirkt wie ein Filter. Aus einer gegebenen Relation werden alle Tupel herausgesucht, welche einer bestimmten Bedingung entsprechen.

Eine *Projektion* ermöglicht die Auswahl bestimmter Spalten einer Relation.

Relationen über gleiche Attributmengen können mit den üblichen Mengenoperationen Vereinigung, Durchschnitt und Differenz verknüpft werden. Dazu müssen die zu verknüpfenden Relationen die gleiche Anzahl Attribute besitzen, und die korrespondierenden Attribute, d.h. die Attribute mit denselben Namen, müssen denselben Wertebereich besitzen.

Das *kartesische Produkt* R X S zweier Relationen R und S ergibt eine Relation mit allen Attributen aus R und S. Identische Attribute in R und S werden in der Ergebnisrelation doppelt als eigenständige Attribute aufgeführt. Als Tupel enthält R X S alle möglichen Kombinationen der Tupel aus R mit den denen aus S.

Der *natürliche Verbund* verschmilzt zwei Relationen bezüglich gleicher Attribute (gleicher Name und gleicher Wertebereich). Der natürliche Verbund R ⊗ S zweier Relationen R und S besitzt als Attribute alle Attribute von R und S, wobei die gemeinsamen Attribute nur einmal aufgeführt werden. Die Tupel von R ⊗ S umfassen alle Kombinationen von R und S, die auf den gemeinsamen Attributen identische Wertekombinationen besitzen.

Beim *Q-Verbund* R[A Θ B]S, wobei Θ einer der Vergleichoperationen =, ≠, ≤, ≥, < oder > ist, wird jedes Tupel aus R mit den Tupeln aus S verknüpft, so daß der Wert des Attributes A der Tupels aus R dem Vergleich Θ mit dem Wert des Attributs B jedes Tupels aus S genügt. Dabei müssen die Wertebereiche der Attribute A und B identisch sein.

Ist die Θ-Vergleichsoperation ein „=„ so spricht man von einem *equi-join*. Dieser entspricht dem natürlichen Verbund, wobei die verglichenen Spalten im Gegensatz zum natürlichen Verbund doppelt aufgeführt sind.

4.2 Transformation

Für die Transformation der Struktur eines Entity-Relationship-Modells - vor allem der verschiedenen Arten von Beziehungen - in Relationen eines relationales Modells gibt es einige Regeln ([BCN92]).

Für die Abbildung der Struktur eines ER-Modells in den Inhalt einer Relation wird die Einführung von Fremdschlüsseln benötigt. Ein *Fremdschlüssel* ist eine Teilmenge der Werte eines Primärschlüssels einer anderen Relation (in diesem Zusammenhang meist *Detailrelation* genannt; die Relation welche den verwendeten Primärschlüssel enthält wird dann als *Hauptrelation* bezeichnet). Oft dient dieser Fremdschlüssel gleichzeitig als Primärschlüssel in der Detailrelation.

Bei der Transformation entsteht aus jeder Entität des ER-Modells eine Relation im relationalen Modell mit den entsprechenden Attributen. Enthält dabei eine Entität mehrwertige Attribute, müssen diese vor der Transformation in eine eigene Entität überführt werden. Bei der Transformation muß außerdem einer

der Schlüsselkandidaten einer Entität des ER-Modells als Primärschlüssel für die Relation des relationalen Modells ausgewählt werden.

Die verschiedenen Arten von Beziehungen können auf unterschiedliche Art transformiert werden. Eine Auswahl der Möglichkeiten soll im Folgenden aufgezeigt werden. Dabei wird davon ausgegangen, daß für die Entität des ER-Modells kein Primärschlüssel festgelegt wurde, sondern daß dieser erst bei der Transformation ausgewählt wird. Werden überwiegend künstliche Primärschlüssel bei der Entwicklung verwendet, ist dies das übliche Vorgehen. In den Beispielen ist jeweils links das ER-Modell und rechts das relationale Modell dargestellt.

- 1:1-Assoziation

Eine 1:1-Assoziation wird am einfachsten in eine Relation umgesetzt, welche die Attribute der beteiligten Entitäten und der Beziehung enthält (Abbildung 4-4).

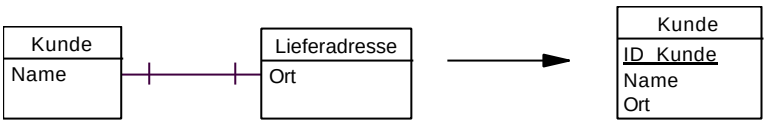


Abbildung 4-4 Transformation einer 1:1-Assoziation

- 1:C-Assoziation

Da bei eine 1:C-Assoziation nicht in jedem Fall eine Beziehung zwischen den beteiligten Entitäten besteht, würden bei einer Umsetzung in eine Relation unter Umständen in vielen Zeilen Attribute ohne Wert bleiben. Deshalb wird hier die Transformation in zwei Relationen bevorzugt, wie in Abbildung 4-5 dargestellt.

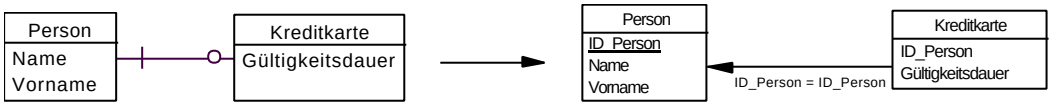


Abbildung 4-5 Transformation einer 1:C-Assoziation

Dabei wird der Primärschlüssel der Relation der Entität mit der Kardinalität 1 (Detailrelation) als Fremdschlüssel in die Relation der Entität mit der Kardinalität C (Hauptrelation) eingetragen. Da der Fremdschlüssel in der Detailrelation auch gleichzeitig ihr Primärschlüssel ist, wird sichergestellt, daß für jeden Datensatz aus der Hauptrelation maximal ein Datensatz in der Detailrelation existieren kann. Die Relation der Entität mit der Kardinalität C enthält auch die eventuell vorhandenen Attribute der Beziehung.

Denkbar ist auch eine Umsetzung in drei Relationen wie bei 1:CM-Assoziation, da die 1:C-Assoziation eine Sonderfall davon ist. Dies ist besonders vorteilhaft, wenn die Beziehung in beide Richtungen optional ist (C:C-Assoziation).

- 1:M-Assoziation

Aus einer 1:M-Beziehung des ER-Modells entstehen in der Regel zwei Relationen im relationalen Modell (siehe Abbildung 4-6).

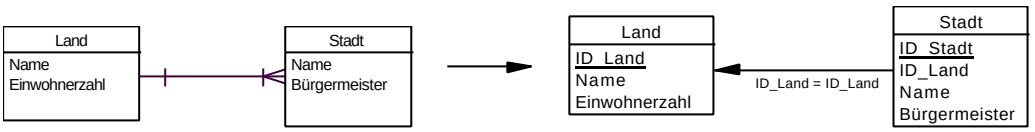


Abbildung 4-6 Transformation einer 1:M-Assoziation

Dabei wird der Primärschlüssel der Relation der Entität mit der Kardinalität 1 (Detailrelation) als Fremdschlüssel in die Relation der Entität mit der Kardinalität M (Hauptrelation) eingetragen. Da der Fremdschlüssel in der Detailrelation nicht ihr Primärschlüssel ist, können für jeden Datensatz aus der Hauptrelation mehrere Datensätze in der Detailrelation existieren. Die Relation der Entität mit der Kardinalität M enthält auch die eventuell vorhandenen Attribute der Beziehung.

- 1:MC-Assoziation

Beziehungen diesen Typs werden am einfachsten wie eine 1:M-Beziehung in Abbildung 4-6 umgesetzt. Da die Relation der Entität mit der Kardinalität MC die Attribute der Beziehung enthält, kann es zu vielen Zeilen mit Attributen ohne Wert kommen, wenn in dieser Hauptrelation viele Datensätze existieren, die keine Beziehung zur Detailrelation besitzen. In diesem Fall ist eine Transformation wie bei einer N:M-Assoziation (Abbildung 4-7) günstiger.

- N:M-Assoziation und N:MC-Assoziation

Zur Umsetzung dieser Assoziationstypen sind immer 3 Relationen notwendig (siehe Abbildung 4-7).

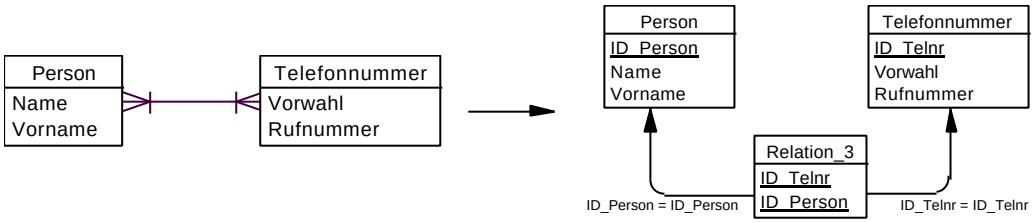


Abbildung 4-7 Transformation einer N:M-Beziehung

Hier entsteht eine Zwischenrelation, welche die Primärschlüssel der beiden anderen Relationen jeweils als Fremdschlüssel enthält. Die beiden Femdschlüsselattribute bilden den zusammengesetzten Primärschlüssel der Zwischenrelation, so daß eine identische Beziehung nicht mehrfach vorkommen kann. Diese Zwischenrelation kann auch die Attribute der Beziehung aufnehmen.

- Generalisierung

Die Transformation einer Generalisierung ist von der Art der Generalisierung abhängig. Die Umsetzung, welche in Abbildung 4-8 zu sehen ist, kann für totale/disjunkte aber auch für partielle/disjunkte Generalisierungen eingesetzt werden. Diese Umsetzung nennt man *alternativen Verweis*([KON97]).

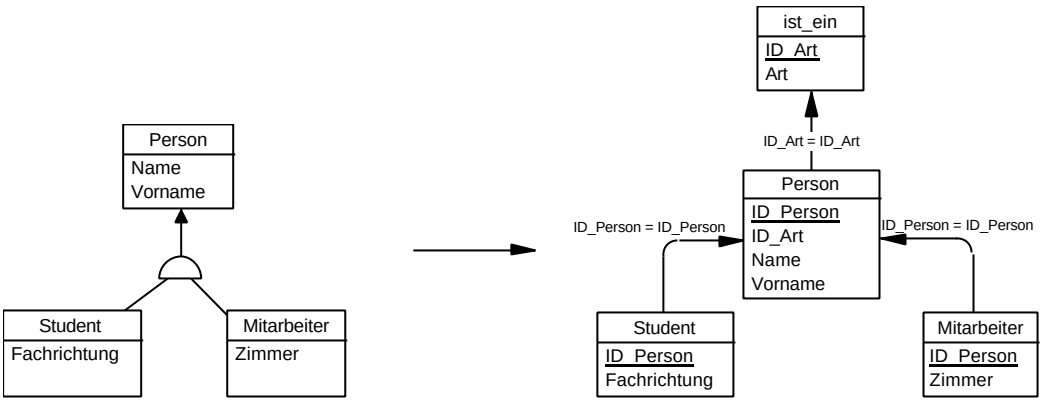


Abbildung 4-8 Transformation einer disjunkten Generalisierung

Die Relationen der Subentiäten enthalten den Primärschlüssel der Superentität als Fremdschlüssel und gleichzeitig als Primärschlüssel. Die *Verweisrelation* ist_ein gibt die Relation der Subentität an, welche die erweiterten Attribute der Superentität enthält. Dabei ist der Primärschlüssel der Verweisrelation als Fremdschlüssel in die Relation der Superentität eingefügt. Kann dieses Fremdschlüsselattribut auch leer bleiben, so stellt dies die Umsetzung einer partiellen Generalisierung, im anderen Fall einer totalen Generalisierung dar.

Zur Transformation einer nicht disjunkten Generalisierung ist eine weitere Zwischenrelation wie in Abbildung 4-9 erforderlich.

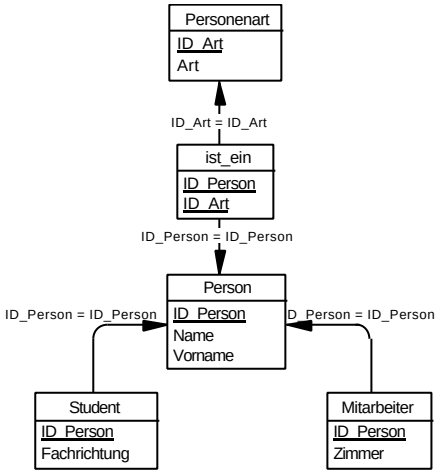


Abbildung 4-9 Transformation einer nicht disjunkten Generalisierung

Dadurch können mehrere Subentitäten Erweiterungen der Superentiät beinhalten.

4.3 Normalisierung

Unter *Normalisierung* versteht man die systematische Untersuchung einer Relation mit dem Zweck, qualitativ hochwertige Relationen zu erhalten. Eine Relation ist dann normalisiert, wenn sie folgende Eigenschaften aufweist:

- sie ist redundanzfrei
- sie verursacht keine Probleme bei der Datenpflege
- sie beschreibt einen Ausschnitt aus der Realität angemessen und richtig

Eines mögliches Problem bei der Datenpflege in einer unnormalisierten (oder nicht ausreichend normalisierten) Relation beschreibt das folgende Beispiel. Eine Relation Auftrag (Auftragnummer., Datum, Betrag, Verkäufernummer., Verkäufer-Name) enthält eine Reihe von Aufträge mit zuständigen Verkäufer. Jeder Verkäufer besitzt dabei eine eindeutige Verkäufernummer. Ändert sich jetzt der Namen eines Verkäufer (z.B. durch Heirat) so muß in allen Aufträgen dieses Verkäufers der Name geändert werden. Wird nun diese Änderung bei einem Auftrag vergessen durchzuführen, so existieren plötzlich zwei Verkäufer mit derselben Verkäufernummer aber unterschiedlichen Namen. Dies kann dann zu erheblichen Problemen führen, wenn z.B. die Aufträge eines bestimmten Verkäufer anhand des Namens des Verkäufers aufgelistet werden. Um solche Probleme zu vermeiden, werden die bei der Transformation aus dem ER-Modell entstandenen Relationen normalisiert.

Unnormalisierte Relationen enthalten nichtatomare Attribute, d.h. die Attribute selbst besitzen eine gewisse Struktur z.B. eine Menge von Werten. Dies kann aber bereits durch eine sorgfältige ER-Modellierung verhindert werden.

- 1. Normalform (1NF)

Eine Relation ist dann in der *ersten Normalform*, wenn alle ihrer Attribute atomar, d.h. ohne Struktur sind.

Eine unnormalisierte Relation kann in die erste Normalform gebracht werden, indem für jedes nichtatomare Attribut eine eigene Relation mit einem Primärschlüssel angelegt wird, und dieser Primärschlüssel in die zu normalisierende Relation anstelle des nichtatomaren Attributs als Fremdschlüssel eingesetzt wird.

- 2. Normalform (2NF)

Eine Relation R in der 1NF befindet sich auch in der *zweiten Normalform*, wenn alle nicht zu einem Schlüsselkandidaten von R gehörenden Attribute von diesem voll funktional abhängig sind.

Um diese Definition verstehen zu können, muß kurz die funktionale Abhängigkeit erklärt werden ([STE93]). Dabei seien X und Y disjunkte Attributkombinationen in einer Relation R.

Y ist von X *funktional abhängig*, wenn zu jedem Zeitpunkt jedem Wert von X genau ein Wert von Y zugeordnet ist. Man schreibt $X \rightarrow Y$.

Besitzt zum Beispiel jeder Verkäufername eine eindeutige Verkäufernummer, so gilt Verkäufernummer $\rightarrow$ Verkäufer, da sich aus der Verkäufernummer der Name des Verkäufers ergibt. Dies gilt allerdings nicht in die andere Richtung. Da es Verkäufer mit gleichem Namen geben kann, kann aus dem Verkäufernamen nicht eindeutig die Verkäufernummer bestimmt werden. Die Verkäufernummer ist also nicht funktional abhängig vom Verkäufername

Die funktionale Abhängigkeit ist übrigens für alle Beziehungen vom Typ 1:1 und M:1 gegeben.

Y ist von X *voll funktional abhängig*, wenn Y von X funktional abhängig ist und von keiner echten Teilmenge von X abhängig ist. Man schreibt $X \bullet\!\!\rightarrow Y$.

Eine Relation in der 1NF kann in die 2NF gebracht werden, indem aus den Attributen, welche bereits von einem Teil des betrachteten Schlüsselkandidaten abhängig sind, eine neue Relation gebildet wird. Die Attribute des betrachteten Schlüssel, von dem die Attribute der neuen Relation abhängig sind, werden als Primärschlüssel für die neue Relation verwendet und fungieren in der zu normalisierenden Relation dann als Fremdschlüssel.

Ein Beispiel soll das Erklärte verdeutlichen. In einer Relation Prüfungsergebnis (Prüfungsnummer, Matrikelnummer, Student_Name, Note) werden Prüfungsergebnisse erfaßt. Als zusammengesetzter Primärschlüssel dient dabei die Prüfungsnummer und die Matrikelnummer, da die Kombination dieser beiden Nummern nur einmal vorkommen kann (vorausgesetzt ein Student kann eine Prüfung nur einmal ablegen). Dabei ist aber der Name des Studenten von der Matrikelnummer, d.h. bereits von einem Teil des Primärschlüssels und damit nicht mehr voll funktional von diesem abhängig. Es muß deshalb eine zweite Relation Student (Matrikelnummer, Student_Name) angelegt und über den Fremdschlüssel eingebunden werden: Prüfungsergebnis (Prüfungsnummer, Matrikelnummer, Note).

- 3. Normalform (3NF)

Eine Relation R in der 2NF befindet sich auch in der *dritten Normalform*, wenn alle nicht zu einem Schlüsselkandidaten von R gehörenden Attribute von diesem nicht transitiv abhängig sind.

Um diese Definition verstehen zu können, soll die *transitive Abhängigkeit* kurz erklärt werden. Dabei seien X, Y und Z disjunkte Attributkombinationen in einer Relation R.

Z ist von X transitiv abhängig, wenn Y von X und Z von Y aber X nicht von Y funktional abhängig ist. Man schreibt dann $X \mapsto Z$.

Um eine Relation in die 3NF zu bringen, müssen die funktionalen Abhängigkeiten zwischen Nicht-Schlüsselattributen, aus welchen solche transitiven Abhängigkeiten entstehen, beseitigt werden. Dies bedeutet, daß aus den transitiv von einem Schlüsselkandidaten (in obiger Definition die Attribute X) abhängigen Attribute (in obiger Definition die Attribute Z) eine neue Relation gebildet wird. Als Primärschlüssel für diese neue Relation werden die Attribute verwendet, von welchen die Attribute der neuen Relation funktional abhängig sind (in obiger Definition die Attribute Y). Diese Attribute werden in der zu normalisierenden Relation auch als Fremdschlüssel verwendet.

In der Relation Auftrag (Auftragnummer., Datum, Betrag, Verkäufernummer., Verkäufer-Name) ist der Verkäufername über die Verkäufernummer von der Auftragsnummer transitiv abhängig, da gilt: Auftragsnummer $\rightarrow$ Verkäufernummer $\rightarrow$ Verkäufer_Name aber die Verkäufernummer vom Verkäufername nicht funktional abhängig ist (wie bereits bei der 2. NF beschrieben). Auch hier wird eine zusätzliche Relation Verkäufer (Verkäufernummer, Verkäufer_Name) angelegt und über den Fremdschlüssel in die zu normalisierende Relation eingebunden: Auftrag (Auftragsnummer, Datum, Betrag, Verkäufernummer).

- Weitere Normalformen

Für die Praxis der Entwicklung einer Datenbasis kaum von Bedeutung sind die weiteren Normalformen Boyce-Codd-Normalform (BCNF), vierte Normalform (4NF), fünfte Normalform (5NF), usw.

Da bei jedem Normalisierungsschritt neue Relationen entstehen, wird das Datenmodell schnell unübersichtlich. Außerdem sinkt die Performance der Datenbankanwendung, da die einzelnen Relationen für viele Abfragen temporär wieder zusammengefügt werden müssen. Deshalb kann an den für die Performance kritischen Stellen auch eine gezielte Denormalisierung stattfinden.

Was im Rahmen der Normalisierung als atomares Attribut bezeichnet wird, hängt letztlich von der Anwendung ab. Werden für die Anwendung z.B. oft Teile eines Attributs wie z.B. der Tag oder das Jahr eines Datums benötigt, so kann es durchaus Sinn machen, statt eines Attributes drei Attribute (Tag, Monat, Jahr) anzulegen. Wird dagegen meist das komplette Datum verarbeitet, so sollte es auch komplett abgespeichert werden.

4.4 Die relationale Datenbank

In den Beispielen zum relationalen Modell wurden immer wieder Beziehungen zwischen Relationen erwähnt, die in der graphischen Form der Darstellung als Pfeil mit Spitze zur Detailrelation angezeigt werden. Nach der Theorie des relationalen Modells existieren keine solchen „festen „ Verbindungen. Die Primär- und Fremdschlüssel (meistens ganze Zahlen) stellen nur Werte der betreffenden Attribute dar. In manchen Datenbanksystemen ist der Anwendungsprogrammierer dafür verantwortlich, daß bei Abfragen die Relationen anhand der richtigen Attribute verknüpft werden, um die innewohnende Struktur für die Dauer der Abfrage sichtbar zu machen.

Andererseits gibt es Datenbanksysteme, welche die Möglichkeit bieten, solche Verbindungen explizit mittels SQL zu implementieren. So kann das Datenbanksystem darüber wachen, daß z.B. in einem Fremdschlüssel nur Werte eingefügt werden, die im Primärschlüssel der Detailrelation vorhanden sind. Dieser Sachverhalt wird *referentielle Integrität* genannt. Werden Tupel, die in einer Haupttabelle referenziert werden, aus der Detailrelation gelöscht oder geändert, so könnte der Fall auftreten, das für den Fremdschlüssel in der Hauptrelation kein Datensatz mit dem entsprechenden Primärschlüssel in der Detailrelation mehr vorhanden ist. Die meisten Datenbanksysteme bieten in diesem Fall drei mögliche Verhaltensweisen an:

- restriced (nicht zulässig):
  Das referenzierte Tupel darf nicht gelöscht und der referenzierte Primärschlüssel darf nicht geändert werden.
- cascade (weitergeben):
  Wird ein referenziertes Tupel gelöscht, werden die Tupel in der Haupttabelle, welche das gelöschte Tupel referenzierten, ebenfalls gelöscht. Wird ein referenzierter Primärschlüssel geändert, so wird der Fremdschlüssel in der Haupttabelle ebenfalls geändert
- set null / set default (auf Null / Vorgabewert setzen):
  Wird ein referenziertes Tupel gelöscht, so wird in den betreffenden Tupeln der Haupttabelle der Fremdschlüssel auf NULL oder einen vorher festgelegten Wert gesetzt.

Manche Sachverhalte aus den Anforderungen können nicht direkt in ein ER-Modell oder in ein relationales Modell eingearbeitet werden, wie z.B. die Beschränkung von Attribute auf bestimmte Werte. Solche Anforderungen müssen mit Hilfe von Integritätsbedingungen implementiert werden. Dazu bieten manche Datenbanksysteme sogenannte Trigger und Datenbankprozeduren an. *Datenbankprozeduren* (auch *stored procedures* genannt). In Datenbankprozeduren werden SQL-Anweisungen mit den Kontrollstrukturen der imperativen strukturierten Programmierung kombiniert. Sie dienen vor allem zur Sicherstellung der Datenkonsistenz unabhängig von den Datenbankanwendungen, da sie zusammen mit der Datenbasis gespeichert werden. Durch ihre Speicherung und Ausführung

innerhalb des Datenbanksystems bieten sie auch einen Geschwindkeitsvorteil bei komplexen Operation mit kleiner Ergebnismenge gegenüber den Funktionen in den Datenbankanwendungen.

Die gleichen Möglichkeiten bieten *Trigger*. Der Unterschied zwischen Triggern und Datenbankprozeduren liegt im Aufruf. Während die Datenbankprozedur von den Datenbankanwendungen explizit aufgerufen werden, werden Trigger durch Ereignisse zwangsweise vom Datenbanksystem gestartet. Solche Ereignisse sind mit den Operationen auf der Datenbasis wie Einfügen, Ändern und Löschen verknüpft und können sowohl vor als auch nach der Ausführung der Operation ausgelöst werden.

Neben den Relationen bzw. Tabellen, welche basierend auf dem relationalen Modell im Datenbanksystem angelegt werden, existieren innerhalb des Datenbanksystems die *Systemtabellen*. Diese enthalten Informationen über Datenbasis, wie Namen, Attribute und Datentypen der einzelnen Tabellen oder Zugriffsrechte. Auf diese Tabellen kann mit entsprechenden Zugriffsrechten durch Anwendungen oder direkt über eine geeignete Schnittstelle bei der Programmierung und Administration zugegriffen werden.

5 SQL

SQL steht für **S**tructured **Q**uery **L**anguage und ist eine 1986 vom American National Standards Institute (ANSI) standardisierte Datenbanksprache. Der 1987 von der International Standards Organisation (ISO) verabschiedete Standard entspricht dem ANSI-Standard weitgehend. 1989 erfolgte eine Revision des Standards. Seither wird der Standard von 1986 als SQL89/Level1 und die verbesserte Version von 1989 als SQL89/Level 2 bezeichnet.

1992 wurde der SQL erweitert. Dieses zur Zeit erhältliche SQL wird als SQL2 oder *SQL-92* bezeichnet. SQL-92 enthält drei verschiedene „Level“. Die bescheidensten Anforderungen stellt der „Entry-Level“, gefolgt von „Intermediate“ und „Full SQL“. Die nächste Fassung wird SQL3 heißen und insbesondere Erweiterungen in Richtung Objektorientierung enthalten ([MAT97]).

Zwischen dem SQL-Standard und den in Datenbanksystemen angebotenen SQL-Implementierungen gibt es Unterschiede in beiden Richtungen. Teilweise übersteigt der verfügbare Sprachumfang die Forderungen der Norm, teilweise werden Forderungen der SQL-Norm nichrt erfüllt. Von vielen Datenbanksystemen wird inzwischen der „Entry-Level“ von SQL-92 erfüllt.

SQL stellt die Schnittstelle zwischen relationalem Datenbanksystem und Anwendungsprogramm dar. Eine Standardisierung von SQL ist wichtig, damit Anwendungen möglichst portabel und herstellerunabhängig sein können, um zum Beispiel aus Gründen der Performancesteigerung oder eines Betriebssystemwechsels auf andere Datenbanksysteme umsteigen zu können.

SQL zählt zu den Sprachen der 4. Generation und ist eine logische, mengenorientierte, nicht-prozedurale Sprache. An Stelle von Anweisungen von Sprachen der 3. Generation (z. B. C, Pascal), die vom Rechner Schritt für Schritt abgearbeitet werden und am Ende das gewünschte Ergebnis erzeugen, tritt eine logische Beschreibung dessen, was als Resultat gewünscht wird. Die Umsetzung dieses Wunsches ist ein Problem des SQL-Interpreters und bleibt dem Anwender verborgen. Um die Anweisungen möglichst effizient umzusetzen, werden sogenannte Optimizer eingesetzt. Als Ergebnis einer Abfrage wird immer eine Menge von Tupeln zurückgeliefert.

Die Befehle in SQL lassen sich in drei verschiedene Sprachen aufteilen:

- DDL-Befehle (**D**ata-**D**escription-**L**anguage)
  Diese Befehle dienen der Erzeugung, Veränderung oder Entfernung von Datenstrukturen (Tabelle, Views, Indizes).

- DCL-Befehle (**D**ata-**C**ontrol-**L**anguage)
  Mit diesen Befehlen werden Zugriffsrechte vergeben.

- DML-Befehle (**D**ata-**M**anipulation-**L**anguage)
  Diese Befehle dienen der Abfrage und Veränderung der Daten.

In Tabelle 5-1 sind die wichtigsten Befehle aufgeführt.

Befehl	Aufgabe		Befehl	Aufgabe
create table DDL	Tabellen erzeugen		insert DML	Zeilen in eine Tabelle einfügen
delete DML	Zeilen einer Tabelle löschen		select DML	Tabellen abfragen
drop table DDL	Tabellen löschen		revoke DCL	Zugriffsrechte entziehen
grant DCL	Zugriffsrechte gewähren		update DML	Daten in einer Tabelle verändern

Tabelle 5-1 SQL-Befehle

Die Funktion eines SQL-Befehls soll nun anhand des SELECT-Befehls erklärt werden.

Abbildung 5-1 zeigt die Syntax des SELECT-Befehls:

```
SELECT [ALL|DISTINCT] {spalten|*}
FROM tabelle [alias] [,tabelle[alias]]...
[WHERE {bedingung|subquery}]
[GROUP BY spalten [HAVING {bedingung|subquery}]]
[ORDER BY spalten [ASC|DESC]... ];
```

Abbildung 5-1 Syntax des SELECT-Befehls

Ein SELECT-Befehl, der alle Klauseln enthält, ist in Abbildung 5-2 dargestellt:

```
SELECT autor, sum(ausleihzahl)
FROM        bücher
WHERE       ausleihzahl > 5
GROUP BY    autor
HAVING      sum(ausleihzahl) > 10
ORDER BY    autor;
```

Abbildung 5-2 Beispiel eines SELECT-Befehls

Dieser Befehl bewirkt, daß aus der Tabelle bücher, welche die Anzahl Verleihvorgänge mehrerer Bücher verschiedener Autoren (Attribute autor, titel, ausleihzahl) enthält, diejenigen Autoren - zusammen mit der Summe der Verleihvorgänge, der von ihnen verfaßten Bücher - ausgegeben werden, deren Büchern

zusammen mehr als 10 Verleihvorgänge aufweisen, wobei nur Bücher mit mehr als 5 Verleihvorgängen berücksichtigt werden. Die Ausgabe wird noch alphabetisch nach Autoren sortiert.
Werden in der FROM-Klausel mehrere Tabellen angegeben, wird das kartesische Produkt der Tabellen gebildet und die weiteren Klauseln darauf angewandt.

Das Zustandekommen des Ergebnisses der Abfrage läßt sich veranschaulichen, wenn man sich vorstellt, daß die einzelnen Klauseln nacheinander ausgeführt werden und jeweils auf dem Ergebnis der vorhergehenden Klauseln angewandt werden (ein realer SQL-Interpreter arbeitet nicht auf diese ineffiziente Art). Die einzelnen Bearbeitungsschritte sehen dann folgendermaßen aus:

1. from bilde das kartes. Produkt der angegebenen Tabellen
2. where streiche alle Zeilen, welche die Bedingung nicht erfüllen
3. group by führe Gruppierung durch (Zusammenfassung der Datensätze, die in den angegeben Attributen den gleichen Inhalt besitzen)
4. having streiche alle Zeilen, welche die Bedingung nicht erfüllen
5. order by sortiere
6. select streiche alle Spalten, die nicht genannt wurden.

Die Sprache SQL ist auf die Abfrage und Manipulation von relationalen Datenbanksystemen zugeschnitten. Daher kann nicht jeder beliebige Algorithmus, wie z. B. in Sprachen der 3. Generation, realisiert werden. Dazu wären Kontrollstrukturen wie Schleife und Verzweigung nötig. Verzweigungen können zwar mit einigem Aufwand erzeugt werden (durch select-Anweisungen, die nur ausgeführt werden, wenn die Bedingungen in den where-Klauseln erfüllt sind) , aber es fehlt jegliche Möglichkeit, eine Schleife oder eine gleichwertige Rekursion zu implementieren. SQL kann deshalb in Sprachen der 3. Generation oder Skriptsprachen von Entwicklungsumgebungen eingebettet werden. Man spricht hier von *embedded SQL*. Die Sprache, in welche die SQL-Anweisungen eingebettet werden, nennt man *Wirtssprache* (host language). Der Austausch von Daten zwischen der Wirtssprache und den SQL-Anweisungen erfolgt durch Variablen der Wirtssprache, welche in den SQL-Anweisungen verwendet werden. In Abbildung 5-3 ist dazu ein kleines Beispiel dargestellt:

```
SELECT      count(*)
INTO        :li_anzahl
FROM        bücher;
```

Abbildung 5-3 Beispiel embedded SQL

Diese Anweisunge speichert die Anzahl (count(*)) der Datensätze der Tabelle bücher in der Variable li_anzahl der Wirtsstprache.

Zur Beschleunigung von Abfragen und zur Sicherstellung der Integrität der Datenbasis kann für eine oder mehrere Attribute einer Tabelle ein Index definiert werden. Es läßt sich dabei festlegen, ob gleiche Werte für einen Index erlaubt sind. Indexe beschleunigen die Abfrage, da das DBMS nicht die gesamte Tabelle nach den gesuchten Daten absuchen muß, sondern über den in einer besonderen Hierarchie gespeicherten Index schneller fündig wird (falls die Daten vorhanden sind). Sind keine gleichen Werte für einen Index erlaubt, wird dies beim Einfügen und Ändern der Daten geprüft. Außerdem kann ein Index zur Sortierung der Daten in einer Tabelle verwendet werden. Ein Index beschleunigt zwar die Abfrage (Leseoperation), allerdings sind Schreibzugriffe (z. B. Einfügen von Daten) durch die größere Anzahl notwendiger Änderungen (Daten, Index) langsamer. Als Index dient meistens der Primärschlüssel.

Mit SQL können die in Kapitel 4.1 aufgeführten Operationen auf Tabellen ausgeführt werden. Dabei entsteht als Ergebnis stets eine (eventuell leere) Menge von Tupeln, die als (temporäre) Tabelle aufgefaßt werden können. Diese Operationen finden bei der Abfrage von Daten Verwendung und werden deshalb vor allem mit Hilfe des SELECT-Befehls durchgeführt.

Die *Selektion* findet durch die WHERE-Klausel des SELECT-Befehls statt. Die Anweisung in Abbildung 5-4 liefert als Ergebnis die Zeilen der Tabelle Bücher, die der Bedingung Seitenzahl < 40 genügen. Dabei werden alle (select *) Attribute dargestellt.

```
SELECT      *
FROM        bücher
WHERE seitenzahl < 40;
```

Abbildung 5-4 Beispiel Selektion

Eine *Projektion* durch die Angabe der gewünschten Attribute in der SELECT-Klausel statt. Die Anweisung in Abbildung 5-5 liefert als Ergebnis die Werte der Attribute Titel und Seitenzahl aller Zeilen der Tabelle Bücher. Der optionale Zusatz DISTINCT erzwingt, daß mehrfach vorkommende Tupel nur einmal aufgeführt werden.

```
SELECT DISTINCT     titel, seitenzahl
FROM                bücher;
```

Abbildung 5-5 Beispiel Projektion

Mengenoperationen finden durch die Verknüpfung mehrerer SELECT-Anweisungen statt. Bei der Mengenoperation Vereinigung findet dabei der Verknüpfungsoperator UNION Verwendung. Die Bildung des Durchschnitts erfolgt durch INTERSECT, die Bildeung der Differenz durch MINUS.

Das *kartesische Produkt* zweier Tabellen findet dadurch statt, daß in der FROM-Klausel des SELECT-Befehls mehrere Tabellen aufgeführt werden. Diese können auch identisch sein. Die Anweisung in Abbildung 5-6 zeigt eine solche Operation. Abbildung 5-7 zeigt das Ergenis.

```
SELECT      *
FROM        R, S;
```

Abbildung 5-6 Beispiel kartesisches Produkt

R	A	B
	a	b
	b	b

S	C	D	E
	1	2	3
	4	5	6
	7	8	9

R X S	A	B	C	D	E
	a	b	1	2	3
	a	b	4	5	6
	a	b	7	8	9
	b	b	1	2	3
	b	b	4	5	6
	b	b	7	8	9

Abbildung 5-7 Ergebnis kartesisches Produkt

Der *natürliche Verbund* kann in SQL nicht direkt angewandt werden. Es muß dagegen der *Q-Verbund* eingesetzt werden. Auch hier findet der Verbund durch die Auflistung mehrerer Tabellen in der FROM-Klausel der SELECT-Anweisung statt. Die zu vergleichenden Attribute und die Vergleichsoperation werden in der WHERE-Klausel angegeben. Sollen die identischen Vergleichsattribute nicht doppelt aufgeführt werden, so muß dies bei der Auflistung der anzuzeigenden Attribute in der SELECT-Klausel berücksichtigt werden. Die Anweisung in Abbildung 5-8 zeigt einen Θ-Verbund, wobei „≤„ als Vergleichsoperation zum Einsatz kommt.

```
SELECT      *
FROM        R, S
WHERE B <= C;
```

Abbildung 5-8 Beispiel Θ-Verbund

R	A	B
	a	b
	c	d

S	C	D
	b	d
	a	b
	d	e

R[B≤C]S	A	B	C	D
	a	b	b	d
	a	b	d	e
	c	d	d	e
	b	b	7	8

Abbildung 5-9 Ergebnis Θ-Verbund

Literaturverzeichnis

[BAL96] Helmut Balzert: *Lehrbuch der Software-Technik: Software-Entwicklung*. Spektrum Akad. Verlag, Heidelberg, 1996.

[BCN92] Carlo Batini, Stefano Ceri, Shamkant Navathe: *Conceptual Databse Design: an entity-relationship approach*. Benjamin/Cummings, Redwood City, California [u.a.], 1992, ISBN 0-8053-0244-1.

[CHE91] Peter P. S. Chen, Heinz-Dieter Knöll: *Der Entity-Relationship-Ansatz zum logischen Systementwurf: Datenbank- und Programmentwurf*. BI-Wiss.-Verlag, Mannheim; Wien; Zürich, 1991, ISBN 3-411-15311-3.

[DAT95] Chris Date: *An Introduction to Database Systems.* Addison-Wesley, Reading, Massachusetts [u.a.], 1995, ISBN 0-201-82458-2.

[KON97] Jens Konnertz: *Konzeption und Implementierung einer relationalen Datenbank unter Verwendung von ER-Modellierung und Benutzerschnittstelle PowerBuilder*. Pflichtstudienarbeit an der Universität Stuttgart, Fakultät Elektrotechnik, Institut für Automatisierungs- und Softwaretechnik, 1997.

[LUD97] Ludewig: *Grundlagen des Software Engineerings*. Skript zur Vorlesung an der Universität Stuttgart, Fakultät Informatik, 1997.

[MAR94] J. Marsch, J. Fritze: *SQL: eine praxisorientierte Einführung*, 2. Auflage, Vieweg, Braunschweig; Wiesbaden, 1994, ISBN 3-528-15210-9.

[MAT97] Günther Matthiesen: *Relationale Datenbanken und SQL: Konzepte der Entwicklung und Anwendung*. Addison-Wesley-Longman, Bonn; Reading, Massachusetts [u.a.], 1997, ISBN 3-8273-1167-5.

[MDA97] Alfred Moos, Gerhard Daues: *Datenbank-Engineering: Analyse, Entwurf und Implementierung relationaler Datenbanken mit SQL*. 2. Auflage, Vieweg, Braunschweig; Wiesbaden, 1997, ISBN 3-528-15183-8.

[RLE97] Andreas Reuter, Frank Leymann: *Grundlagen der Datenbanken und Transaktionssysteme.* Skript zur Vorlesung an der Universität Stuttgart, Fakultät Informatik, 1997.

[STE93] Gerhard Stegemann: *Datenbanksysteme: Konzepte, Modelle, Netzanwendung*. Vieweg, Braunschweig; Wiesbaden, 1997, ISBN 3-528-04935-9.

