Hochschule München
Fachbereich Informatik

Datenmanagement SS2008

Ulrike Gröttrup
Dr. Thomas Tensi

Inhaltsverzeichnis

Information

Das vorliegende Dokument ist Grundlage unserer Vorlesung "Datenmanagement" im SS2008 an der Hochschule München. Da es sich um eine Arbeitsunterlage handelt, enthält das Papier möglicherweise Fehler und in jedem Fall sind Diagramme enthalten, die aus im Literaturverzeichnis zitierten Büchern stammen. Vor einer weiteren Verwendung des Materials muß daher geprüft werden, ob Rechte Dritter damit verletzt werden.

München im April 2008

Ulrike Gröttrup und Thomas Tensi

Kapitel 1

Datenbanken — grundlegende Konzepte

1.1 Motivation

Eine *Datenbank (DB)* ist zunächst einmal eine Ansammlung von Daten, wobei man den Begriff üblicherweise für auf Rechnern gespeicherte Daten verwendet. Damit man mit den Daten etwas anfangen kann, braucht man eine Software, die auf die Daten zugreift und sie nutzbar macht. In vielen Fällen tut man dies mit speziellen Programmen, die auf spezielle Datenformate zugreifen, z.B.

- ein Textverarbeitungsprogramm auf einen Text
- ein Dateiverwaltungsprogramm auf Dateien
- ein Adressverwaltungssystem auf Adressen
- eine Lagerverwaltung auf Artikel, Bestände
- ein Bestellsystem auf Artikel, Kunden und Bestellungen

Die Nutzung dieser Daten setzt einige grundlegende Funktionen voraus, die unabhängig vom Geschäftszweck sind, d.h. unabhängig von der Art der Anwendung, die mit den Daten arbeiten soll:

- Daten speichern,
- Daten lesen,

- Daten suchen und finden,
- Daten verändern.

Diese Funktionen sind in einfacher Form bereits im Betriebssystem des Rechners realisiert, z. B. für Dateien als nicht weiter beschriebene Bytefolgen. Sobald die Daten eine Struktur erkennen lassen, ergibt sich der Bedarf, diese Struktur der Daten als für den Umgang damit zu nutzen. Ich möchte z.B. die Adresse eines Kunden nicht aus einem fließenden Text heraussuchen, sondern anhand bestimmter Ordnungskriterien finden. Damit das geht, ist es notwendig, die Struktur der Daten zu beschreiben und die Daten dann strukturiert zu speichern.

Ein System aus Daten und dazu gehörigen Funktionen ist ein *Datenbanksystem (DBS)*. Jedes solche System muss eine Reihe von Anforderungen erfüllen, die je nach Anwendungszweck unterschiedlich gewichtet sind, aber prinzipiell immer wieder auftauchen:

- Speicherökonomie (Besonders bei großen Datenmengen relevant)
- Strukturabbildung (Dinge der Geschäftswelt wiederfnden)
- Sortierung (Listen erzeugen können)
- Schnelle Lese-/Schreibzugriffe
- Nutzbarkeit durch unterschiedliche Anwendungen
- Widerspruchsfreiheit (Bedingungen an Daten einhalten)
- Redundanzfreiheit (Daten nicht mehrfach ablegen)
- Mehrnutzerfähigkeit (Daten gleichzeitig bearbeiten)

Die Umsetzung der Basisfunktionen eines jeden Datenbanksystems unter Einhaltung dieser Standardanforderungen ist ebenfalls noch weitgehend unabhängig vom Geschäftszweck einer speziellen Datenbankanwendung und kann daher allgemeingültig in einem Softwaresystem realisiert werden. Ein *Datenbankmanagementsystem (DBMS)* ist ein solches Softwaresystem, das dazu da ist, Datenbanken aufzubauen und nutzbar zu machen.

In einem Datenbankmanagementsystem sind einige grundlegende Probleme gelöst, die der Umgang mit Daten mit sich bringt. Diese Systeme haben Konzepte implementiert, die es dem Nutzer abnehmen,

- Daten sortiert und trotzdem speicherplatzeffizient abzulegen,
- Die Struktur der Daten abzubilden
- Optimierte Suchzugriffe auszuführen
- Daten dem Anwendungszweck entsprechend zu optimieren

Ziel dieser Vorlesung ist die Vermittlung der Systematik und Methode, um Daten in eine gut nutzbare Form zu bringen, sowie die grundlegenden Konzepte kennenzulernen, die in Datenbankmanagementsystemen implementiert sind. Zur Einordnung dieser Systeme ist es hilfreich, sich noch einmal die Schichten ins Gedächtnis zu rufen, aus denen Softwaresysteme im Allgemeinen bestehen.

1.2 Schichtenarchitekturen

Die Aufteilung der gesamten Software eines Anwendungssystems in Schichten ist einerseits willkürlich, andererseits aber eine sehr sinnvolle Trennung der verschiedenen Abstraktionsebenen auf einem Rechner. Eine gute, d.h. eindeutige, transparente und konsequente Schichtenarchitektur ist entscheidend für die Qualität des Gesamtsystems.

Ein gängiger Ansatz zur Einteilung in Schichten ist der folgende:

Individual-Software
Anwendungs-Software
Basis-Software
System-Software
Betriebssystem

Softwaresysteme müssen nicht unbedingt nur Teile einer Schicht enthalten, sondern können auch schichtenübergreifend angesiedelt sein.

- Betriebssysteme sind die unterste Softwareebene, in der die Basisfunktionen des Rechners bereitgestellt werden: DOS, Windows, UNIX, VMS, OS/2, ...
- System-Software ist direkt auf das Betriebssystem aufbauende Software, die allgemeine Funktionen anbietet. Dazu gehören z.B. Treiber, Dateiverwaltungssysteme oder Programme zur Organisation der Hardware

- Basis-Software ist die erste Ebene, die sich für die direkte Nutzung anbietet, die über die Organisation des Rechners hinausgeht, wie z.B. Textverarbeitungsprogramme, Graphikprogramme etc.

- Anwendungs-Software ist eine auf einen bestimmten Anwendungszweck zugeschnittene Software, wie z.B. Buchhaltungsprogramme, CAD-Programme oder Büroorganisationssoftware

- Individual-Software ist eine eigens für ein Unternehmen oder einen bestimmten Anwendungszweck in einem Unternehmen entwickelte Software, also das, was als Auftragsentwicklung von der Mehrzahl der Softwarefirmen angeboten wird.

Man sieht sehr schnell, dass diverse Softwaresysteme Funktionsblöcke aus verschiedenen Schichten beinhalten. Windows z.B. stellt außer dem eigentlichen Betriebssystem auch eine Vielzahl von Treibern, also System-Software bereit, dazu Basis-Software wie eine simple Textverarbeitung und ein einfaches Graphikprogramm und sogar auch Anwendungs-Software wie z.B. einige einfache Spiele.

Auch Datenbanksysteme beinhalten immer Anteile der System-Software, da sie direkt auf dem Betriebssystem aufsetzen, darüber hinaus aber auch Teile bis hin zu der Anwendungs- oder Individual-Software. Datenbankmanagementsysteme stellen System-Software bereit (Komponenten zum Datenzugriff), darüber hinaus aber auch Basis-Software (Interaktive Datenbankabfrageprogramme) und Anwendungssoftware (Entwicklungsumgebung für Datenbanken).

1.3 Grundsatzkonzepte

- Redundanzfreiheit, d.h. das Vermeiden von Mehrfachspeicherung (Probleme bei redundanter Speicherung sind Verschwendung von Speicherplatz, komplexe Prozesse bzw. Auseinanderlaufen der Information bei Änderungen)

- Zugriffssprache, d.h. ein programmsprachliches Repertoire von Befehlen für Datenbearbeitung, möglichst standardisiert

- Effizienz bzgl. Speicherplatz und Zugriffszeit, was in Konkurrenz zueinander steht

- Datensicherheit, d.h. sowohl Vermeidung von Datenverlust wie auch unqualifizierter bzw. unauthorisierter Manipulation von Daten

- Zugriffskontrolle, d.h. Verhindern unauthorisierter Datenzugriffe (lesend und schreiben)

- Datenintegration als Gegenkonzept, d.h. Bereitstellung einer einzigen Datenbasis, in der alle Daten redundanzfrei gespeichert sind und in der zentral alle Basisanforderungen an die Datenbank umgesetzt sind, so dass diese beim Zugriff durch beliebige beteiligte Systeme eingehalten werden.

- Transaktionskonzept

1.4 Historische Entwicklung

Beginn der 60er Jahre: anwendungsspezifische Datenorganisation Daten wurden in sequentiellen Dateien abgelegt (oft mit strikt sequentiellen Zugriff bei Magnetbändern). Die Datenorganisation wurde von der Anwendung geräteabhängig vorgenommen.

Mitte der 60er Jahre: Datenverwaltungssysteme, Dateisysteme Abstraktionsschichten zwischen abstrakten Daten und ihrer gerätespezischen Repräsentationen wurden eingeführt (sequentieller und indexsequentieller Dateizugriff). Zusätzlich wurden wesentliche Dienste von diesen Schichten erbracht (Mischen, Sortieren).

Mitte bis Ende der 60er Jahre: erste Datenbankmanagementsysteme Als weitere Abstraktion wurden gängige Strukturen von Datentypen in Datenbankmanagementsystemen abgebildet (Baumstrukturen und Netze, z.B. Honeywell IDS). Wesentlicher Nutzen war die Integration aller Daten in einen Datenpool.

1972: relationale Datenbankmanagementsysteme Basierend auf den Forschungsarbeiten von Codd zum Relationenkalkül wurden erste relationale DBMS entwickelt. Idee war, dass alle Datentypen und ihre Beziehungen als mathematische Relationen auf wenigen Elementartypen modelliert werden. Dieser Ansatz galt damals als nicht effizient implementierbar und war es zunächst auch.

1976: Modellierung über das E/R-Modell Zwischen Dingtypen und ihren Beziehungen wird unterschieden; das ist weniger abstrakt, aber leichter verständlich.

1986: Standardisierung von SQL Als Zugriffssprache auf relationale Datenbanken wird SQL standardisiert, allerdings abweichend von Codds Kalkül.

in den 80er und 90er Jahren: Konsolidierung In den letzten zwei Jahrzehnten wurden praktisch überall relationale Datenbanken eingeführt. Aufgrund der extrem gestiegenen Rechnerleistung und -kapazität zusammen mit modernen logischen und technischen Optimierungen ist die DB-Leistung kaum noch ein Problem, wohl aber die Konsistenz der Daten in großen Unternehmen.

90er Jahre bis heute: Fortschritte bezüglich Abstraktion und Umsetzung Objektorientierte Datenbanken passen besser zu gängigen Programmiersprachen; Verteilung von Datenbanken ist umsetzbar. Große Datenbanken erlauben Anwendungen wie Data Warehouses.

1.5 Struktur von Datenbanksystemen

Abbildung 1.1 zeigt eine Übersicht über die Struktur eines Datenbanksystems.

Es gibt folgende Bestandteile:

Externe Ebene: Schnittstelle nach außen, für den Benutzer sichtbare Schicht

- DB-Operationen: Anfrage- und Änderungs-Operationen auf der Datenbank
- Einbettungsmechanismen: Einbettung von Datenbankoperationen in Anwendungsprogrammen
- Dialogoberfläche: Entwurf und Nutzung von Menus und Masken für die Verwaltung von Daten

Konzeptionelle Ebene: Kern des Datenbanksystems

- Datendefinition: Strukturmodell der Daten aus abstrakter Sicht

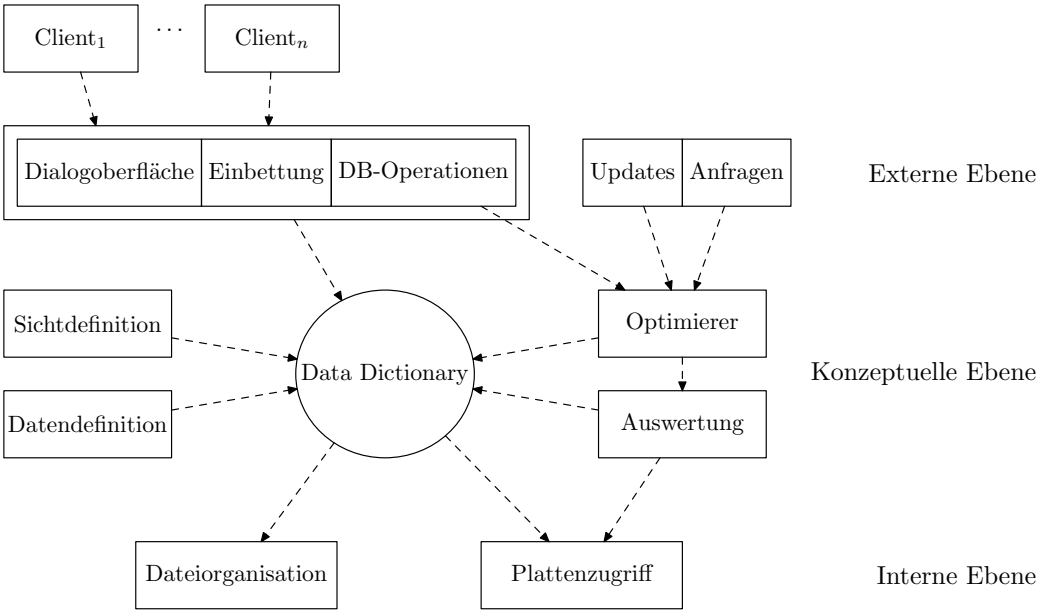


Abbildung 1.1: Übersicht über die Struktur eines Datenbanksystems

Sichtdefinition: Definition von Benutzersichten auf das Strukturmodell

Optimierer und Auswertung: effiziente Umsetzung von Anfragen und Änderungen auf dem Strukturmodell

Interne Ebene: Abbildung auf technische Plattform

Dateiorganisation: Definition der (inneren) Dateiorganisation des DBMS und der Zugriffspfade

Plattenzugriff: technische Organisation des wahlfreien Zugriffs auf Hintergrundspeicher

Kapitel 2

Schemaarchitekturen — Modellierung

2.1 Schema

Zur Einordnung des Begriffes *Schema* bzw. *Datenbankschema* hier noch einmal die unterschiedlichen Ansätze für Architekturen.

2.2 Ansätze zur Architekturbetrachtung

- Die *Schema-Architektur* beschreibt die Struktur der Daten, das Schema gibt also ein formales Modell der Struktur von Daten wieder. Die Schema-Architektur steht im Mittelpunt dieser Vorlesung und wird daher im weiteren vertieft behandelt.

- Die *System-Architektur* eines Datenbanksystems beschreibt dessen Aufbau aus technischen Komponenten. Eine Darstellung der typischen Systemarchitektur eines Datenbanksystems wurde im letzten Kapitel gegeben. Außerdem ist dieser Bereich in der Vorlesung Datenbanksysteme ausführlich besprochen worden.

- Die *Anwendungsarchitektur* beschreibt dagegen den Aufbau des Datenbanksystems aus fachlichen Komponenten, d.h. Benutzerkomponenten und fachlichen Schnittstellen. Teilweise wird darunter auch das dynamische Modell aller Abläufe in der Gesamtanwendung verstanden.

2.3 Schema-Architektur

Ein Schema, also ein formales Modell der betrachteten Daten, spielt in Datenbanksystemen gleich in verschiedenen Kontexten bzw. aus verschiedenen Sichten eine Rolle.

- Das *konzeptuelle Schema* beschreibt die Struktur der Daten im logischen Sinn, und zwar auf den verschiedenen denkbaren Abstraktionsebenen. Diese sind die fachliche Datenmodellierung, der technische Datenbankentwurf und schließlich die eigentliche Definition der Daten in einer Datenbank. Das konzeptuelle Schema ist an sich unabhängig von Systemvoraussetzungen und gewählter Implementierung.
- Das *interne Schema* dagegen ist eine Darstellung der Datenorganisation, d.h. der Systematik, mit der die Daten tatsächlich im Dateisystem abgelegt werden. Teil des internen Schemas ist die Bereitstellung der Zugriffspfade. Damit ist das interne Schema abhängig von der jeweiligen Systemumgebung sowie den verwendeten Sprachkonzepten. Mit diesem Teil der Schemadefinition werden wir uns im Laufe der Vorlesung nicht vertieft befassen.
- Das *externe Schema* ist eine spezielle Sicht auf ein konzeptuelles Schema, die einem bestimmten Anwendungszweck angepasst ist. Das bedeutet, dass es mehrere externe Schemata auf ein und dieselben Daten (Datenmodelle) gibt, die dem jeweiligen Anwendungszweck angepasst sind.

Vorteile der Trennung in die verschiedenen Schemata bei der Verarbeitung von Änderungen:

- Das *konzeptuelle Schema* bleibt unabhängig von Änderungen in der internen Dateiablage gültige Basis für die Sichtendefinition nach extern. (Stabilität der Benutzerschnittstelle)
- Grundlegende Eigenschaften des *internen Schemas* können beibehalten werden bei Änderungen des konzeptuellen Schemas und erst recht der Sichten, allerdings schlagen die meisten Änderungen durch.
- Das *externe Schema* kann gegenüber Erweiterungen des konzeptuellen Schemas unverändert bleiben, solange die Sicht, für die das externe Schema gilt, die neuen Daten nicht benötigt. Änderungen schlagen genau dann durch, wenn auch die Sicht betroffen ist. Sobald das externe

Schema von Änderungen betroffen ist, sind in der Regel auch die Anwendungsprograme betroffen, die auf diesen Sichten operieren.

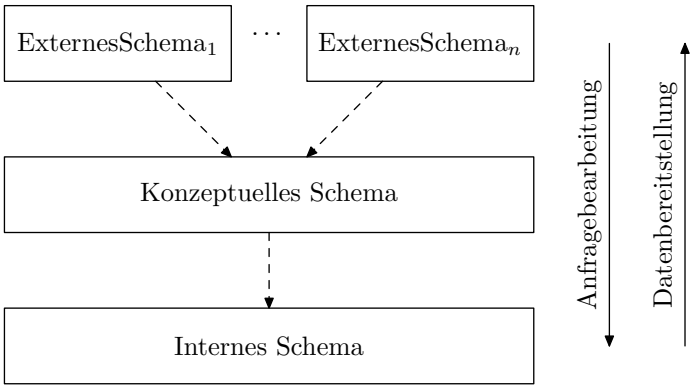


Abbildung 2.1: Drei-Ebenen-Schema-Architektur für Datenbanken

2.4 Beispiel für die drei Schema-Ebenen

konzeptuelles Schema in Relationendarstellung

AUTOR

Name	Vorname	Nr	BuchID → BUCH.BuchID
Meier	Max	1	4711
Schulze	Moritz	1	0815
Ibsen	Uli	2	0815
Müller	Irene	1	1234
...	...	...	...

BUCH

BuchID	Titel	Jahr	ISBN
4711	Datenbasen I	1993	3-452-12
0815	UNIX X	1998	1-424-11
1234	ORACLE	1992	1-123-12
...	...	...	...

Externe Sicht in Relationendarstellung Eine mögliche externe Sicht auf die Daten wäre die simple Betrachtung der Liste aller Buchtitel, ohne besondere Berücksichtigung der pro Autor bekannten Informationen.

TITEL

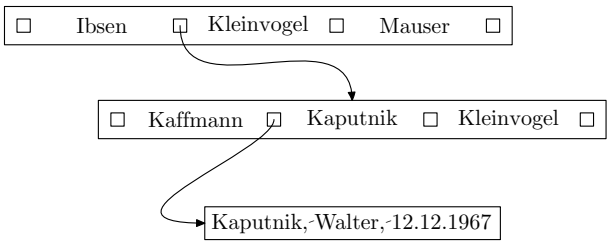
Titel	BuchID	Jahr	ISBN	Name → AUTOR.Name
Datenbasen I	4711	1993	3-452-12	Meier
UNIX X	0815	1998	1-424-11	Schulze
UNIX X	0815	1998	1-424-11	Ibsen
ORACLE	1234	1992	1-123-12	Müller
...	...	...	...	...

Externe Sicht in hierarchischer Darstellung

TITEL

Titel	BuchID	Jahr	ISBN	Name → AUTOR.Name
Datenbasen I	4711	1993	3-452-12	Meier
UNIX X	0815	1998	1-424-11	Schulze
				Ibsen
ORACLE	1234	1992	1-123-12	Müller
...	...	...	...	...

Interne Sicht auf die Daten Hier ist die Zugriffsstrategie auf die Daten erkennbar, z.B. für eine Baumstruktur wie in Bild 2.2a oder über eine Hash-Tabelle wie in Bild 2.2b.



(a)

Hashcode	Buchtitel
...	...
123	UNIX X
124	
125	Datenbasen 1
...	...

(b)

Abbildung 2.2: Anordnung der Personentabelle als B-Baum (a) und der Buchtitel als Hashtabelle (b)

2.5 Instanziierungsebenen

Eine Instanz ist ein „Exemplar“ oder eine „Ausprägung“ eines gegebenen Schemas. Unter *Instanziierung* versteht man also die Belegung eines Modelles mit konkreten Werten. Man unterscheidet unterschiedliche Instanziierungsebenen daher entsprechend der Belegung mit konkreten Ausprägungen:

- Das *Schema* ist das abstrakte Modellebene eines Datenbestandes (KUNDE mit Name, Adresse, ...).
- Ein bestimmter Satz von Anwenderdaten ist eine *Instanz* des Schemas (KUNDE “Meier”, „Parkstraße 11, Monopolystadt“’).
- Umgekehrt gibt eine *Modellierungssprache* eine Abstraktion von Modellen (Schemata) vor, die durch ein konkretes Schema instanziiert wird.

Kapitel 3

Datenbankmodelle

Definition 3.1 (Datenbankmodell)
Ein *Datenbankmodell* ist ein System von Konzepten zur Beschreibung von Datenbanken. Es legt Syntax und Semantik von Datenbankbeschreibungen für ein Datenbanksystem fest. □

Die Datenbankbeschreibung wird in der Regel als *Datenbankschema* bezeichnet.

Bestandteile eines Datenbankmodells sind

statische Eigenschaften der Daten, d.h.

- die *Objekte*, d.h. eine Klassifikation der Dinge und ihrer Eigenschaften bis hin zu den verwendeten Datentypen und
- die *Beziehungen* zwischen den Objekten, d.h. eine Beschreibung der Zusammenhänge und Abhängigkeiten dieser Objekte untereinander

dynamische Eigenschaften der Daten, wie

- die *Operationen*, errechenbarer Eigenschaften der Objekte und
- *Beziehungen* zwischen Operationen, also Aussagen über dynamische Abhängigkeiten

Integritätsbedingungen , d.h. logische Aussagen über die

- *Objekte* und
- *Operationen*,

die zu jeder Zeit gültig sein müssen und deren Einhaltung innerhalb der konkret vorliegenden Daten zu jedem Zeitpunkt sichergestellt werden muss.

3.1 Grundlagen von Entity-Relationship-Modellen

Der Begriff des Entity-Relationship-Modells geht zurück auf einen Artikel von P.P. Chen 1976.

3.1.1 Grundkonzepte

Die drei grundlegenden Konzepte sind folgende:

entity (Entität): Typ von Objekten der realen oder Vorstellungswelt, die für das Informationssystem bedeutsam sind (z.B. ein Buch, eine Person...);

relationship (Beziehung): Beziehung zwischen Entitäten (z.B. eine Person ist Autor eines Buchs);

Attribut: datenhaltige Eigenschaft von Entitäten oder Beziehungen (z.B. das Erscheinungsdatum eines Buchs);

Achtung: man kann auch ein Objekt als "Entität" ansehen und die Abstraktion als "Entitätstyp" bezeichnen! Wir weichen von diesem Sprachgebrauch ab, da im Kern der E/R-Modellierung nicht konkrete Objekte, sondern deren Abstraktionen stehen. Auch in der Klassenmodellierung der UML werden Klassen und nicht Objekte modelliert.

Die obigen Konzepte werden im E/R-Modell wie folgt graphisch dargestellt:

- Entitäten durch ihren Namen in einem Rechteck (siehe beispielsweise "Professor" in Bild 3.1);
- Relationen durch ihren Namen in einer Raute mit Verbindungslinien zu sämtliche beteiligten Entitäten (siehe beispielsweise "liest" in Bild 3.1);
- Attribute durch ihren Namen in abgerundeten Rechtecke verbunden mit der übergeordneten Entität; wenn der Datentyp des Attributs bekannt ist, folgt er dem Attributnamen und einem Doppelpunkt (siehe beispielsweise "Titel" von "Vorlesung" in Bild 3.1).

Wenn ein einzelnes Attribut oder ein Tupel von Attributen jedes Objekt einer Entität eindeutig charakterisieren, dann heißen diese ein *Schlüssel* der Entität. Die Namen der Attribute, die zum Schlüssel gehören, werden in der graphischen Darstellung unterstrichen (siehe beispielsweise "ISBN-Nr." von "Buch"in Bild 3.1).

Auch Beziehungen können Teil eines Schlüssels sein.

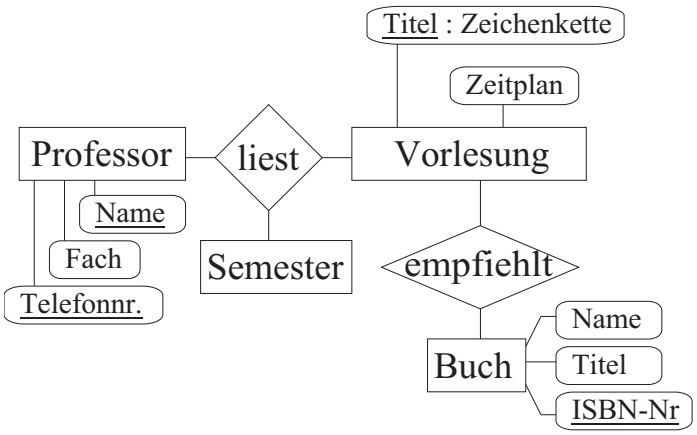


Abbildung 3.1: Beispiel eines ER-Modells

3.1.2 Weitere Elemente der E/R-Modellierung

Die E/R-Modellierung umfasst weitere Notationen, die in verschiedenen Situationen zur Anwendung kommen, in denen Attribute oder Beziehungen spezielle Eigenschaften haben.

Betrachtungen zu Attributen

Bei Attributen einer Entität treten folgende speziellen Konstellationen auf, die im E/R-Modell berücksichtigt werden können.

einfach/zusammengesetzt: Attribute können einfach sein, d.h. aus einem atomaren Typ bestehen oder zusammengesetzt sein aus Bestandteilen mit beliebigem Typ. Atomare Typen sind dabei solche, deren innere Struktur nicht interessiert (Ganzzahl, Datum, Zeichenkette).

Im E/R-Diagramm werden die Bestandteile eines zusammengesetzten Attributs mit diesem verbunden wie mit einer Entität.

abgeleitet: Wenn ein Attribut sich durch Berechnung aus anderen ergibt, dann bezeichnet man es als *abgeleitet* (typisches Beispiel: Alter einer Person).

Im E/R-Diagramm werden abgeleitete Attribute durch gepunktete Ovale notiert.

einwertig/mehrwertig/optional: In der Regel hat ein Attribut genau einen Wert seines Attributtyps, es ist *einwertig*. Kann es mehrere Werte gleichzeitig haben, also eine Menge von Werten seines Typs, ist es *mehrwertig*. Wenn es keinen Wert haben darf, dann ist es *optional* (dafür wird dann in der Datenbank der spezielle Wert NULL verwendet).

Im E/R-Diagramm werden mehrwertige Attribute durch einen Doppelkreis notiert; für optionale gibt es keine graphische Notation.

Primärschlüssel/Sekundärschlüssel: Es kann passieren, dass unterschiedliche einzelne Attribute oder unterschiedliche Tupel von Attributen jedes Objekt einer Entität eindeutig charakterisieren. Alle diese Tupel heißen *Schlüssel* der Entität. Der ausgewählte ist der *Primärschlüssel*, die Alternativen sind *Sekundärschlüssel*.

Im E/R-Diagramm werden Primärschlüsselattribute unterstrichen; für Sekundärschlüsselattribute gibt es keine Notation.

Datentypen Man kann bei der E/R-Modellierung den Typ eines Attributs angeben.

Im E/R-Diagramm folgt der Datentyp dem Namen im Kreis nach einem Doppelpunkt.

Betrachtungen zu Beziehungen

Bei Attributen einer Entität treten folgende speziellen Konstellationen auf, die im E/R-Modell berücksichtigt werden können.

Kardinalität/Optionalität: Die *Kardinalität* einer Beziehung ist eine Quantifizierung der möglichen Teilnahmen von Entitätsinstanzen an einer Beziehung (z.B. die Anzahl der Person, die als Autor eines konkreten Buchs auftreten können (einer bis beliebig viele)).

Im E/R-Diagramm werden Kardinalitäten als Intervallangabe “[m,n]” von natürlichen Zahlen an der Verbindung zur Entität notiert; * ist für “beliebig viele” erlaubt.

Generalisierung/Spezialisierung (IST-Beziehung): Wenn eine Entität ein Spezialfall einer anderen ist (in der Regel mit Zusatzattributen), dann kann man zwischen diesen eine *IST-Beziehung* modellieren (z.B. Buch ist Veröffentlichung, Zeitschrift ist Veröffentlichung usw.)

Im E/R-Diagramm werden IST-Beziehungen statt durch eine Raute auf der Beziehungslinie durch ein auf die generalisierende Entität zeigendes Dreieck notiert.

Beziehungen als Attribute: Wenn eine Entität eine Beziehung zu einer anderen Entität hat, dann kann man die Objekte der anderen Entität als Attributwerte der ersten ansehen. Um dafür auch griffige Attributnamen zu haben, können die Beziehungen mit *Rollennamen* versehen werden (neben der Kardinalitätsangabe).

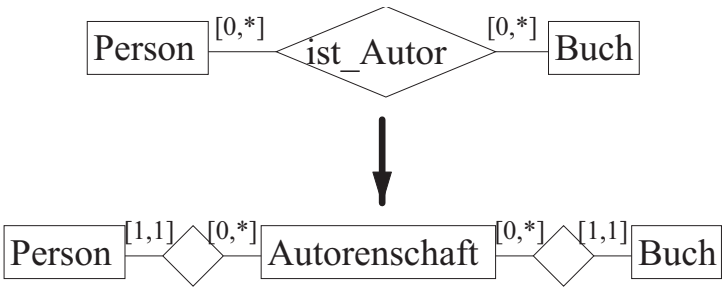
Beispiel: Person ist Autor eines Buchs, dann kann man auch Person als einwertiges strukturiertes Attribut "Autor" von Buch ansehen.

binäre vs. n-äre Beziehungen: Relationen können nicht nur zwischen zwei Entitäten verlaufen, sondern zwischen n (Beispiel: "Person" hat "Termin" in "Raum"). Die Kardinalität bezieht sich auf die Häufigkeit einer Entität in einer Situation, bei der alle Partnerentitäten genau einmal vorkommen (siehe unten).

Im E/R-Diagramm werden alle beteiligten Entitäten mit der Relationsraute verbunden.

m:n-Beziehungen/Auflösung in eigener Entität: Wenn man in einem implementierungsnahen Modell keine m:n-Beziehungen haben will, können diese über eine neue Entität in zwei 1:n-Beziehungen aufgelöst werden.

Beispiel: Person ist Autor von Buch $\rightsquigarrow$ Einführen einer Autorenschaft-Entität



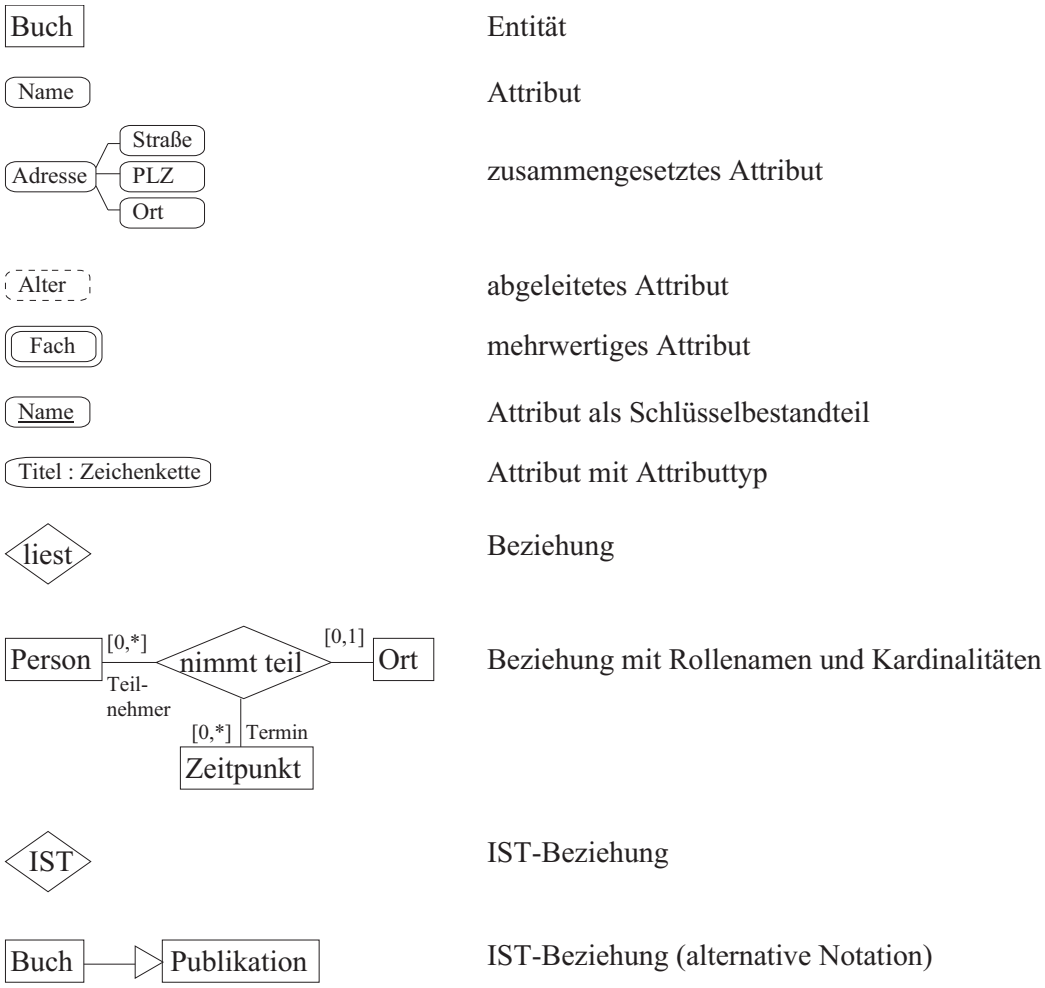


Abbildung 3.2: graphische Notationen für die ER-Modellierung

3.2 Mathematisches Modell der E/R-Modellierung

In diesem Abschnitt beschreiben wir, wie man für die E/R-Modellierung ein mathematisches Modell angeben kann (ein Metamodell der E/R-Modellierung). Es soll lediglich illustrieren, wie bestimmte Begriffe sauber definiert werden; wir werden es in den folgenden Kapiteln nicht weiter verfolgen.

Definition 3.2 (Infrastrukturmodell für das E/R-Metamodell)

Domänen Die Menge D der Domänen ist die Menge von Elementartypen (Ganzzahl, String, Datum, ...) in unserem Modell. Jedes Element ist die Menge aller Ausprägungen eines Elementartyps (also z.B. ist $\mathbb{Z} \in D$!).

Bezeichner Die Menge ID aller Bezeichner enthält Namen, die als Entitäts-, Attribut- oder Rollennamen auftreten dürfen.

Wir gehen davon aus, dass die Menge ID aus Zeichenketten von Buchstaben besteht. Wir schreiben die Namen ohne Anführungszeichen, wenn aus dem Kontext klar ist, dass die Zeichenkette für einen Bezeichner steht.

Typbezeichner Die Menge $TypID \subset ID$ sind die Typbezeichner, also Namen von Elementartypen.

Typfunktion Die Funktion $dom : TypID \to D$ ist die Abbildung von Typbezeichnern in Elementartypen. Beispielsweise könnte $dom(\text{Ganzzahl}) = \{0, -1, 1, 2, -2, \ldots\} = \mathbb{Z}$ sein.

□

Anmerkung: Wir werden im Folgenden gelegentlich die Projektionsfunktion π_i benutzen, die das i-te Element eines Tupels gewinnt.

Definition 3.3 (mathematisches Modell für Entitäten)
Eine Entität ist ein Tupel aus folgenden Teilen:

- eine Menge von Attributnamen $A \subset ID$,
- eine Menge von Schlüsselattributnamen $S \subseteq A$,
- eine Abbildung $attTyp : A \to TypID$, die jedem Attributnamen einen Typ zuordnet,

- eine Abbildung $card : A \to \mathbb{N} \times \mathbb{N} \cup \{\infty\}$, die jedem Attributnamen eine Kardinalität zuordnet

Die Menge aller Entitäten heiße *Ent*. □

Anmerkung: In dem vorgestellten Modell dürfen Attribute als Typ nur Elementartypen haben. Das ließe sich beheben durch entsprechende Erweiterung der Menge *TypID* der Typen.

Beispiel 3.4 (Entität)
Die Entität "Professor" aus unserem Beispiel würde modelliert als Tupel

(A = { Name, Fach, Telefonnummer }, S = { Name },
attTyp = { (Name, Zeichenkette), (Fach, Zeichenkette),
(Telefonnummer, Ganzzahl) },
card = { (Name, (1,1)), (Fach, (0,1)), (Telefonnummer, (0,1)) })

Definition 3.5 (Objekt)
Ein Objekt besteht aus folgenden Teilen:

- eine Menge von Attributnamen A,
- eine Abbildung $value : A \to \wp\left(\bigcup_{i \in D} i\right)$,
  (d.h. Attributnamen werden auf Mengen von Werten abgebildet)

Die Menge aller Objekte heiße *Object*. □

Definition 3.6 (Objekt als Instanz einer Entität)
Ein *Objekt o ist Instanz einer Entität e* (i.Z. *o* **instanziiert** *e*), gdw. folgendes gilt:

- $o.A \supseteq e.A$,
- für_alle $a \in e.A : o.value(a) \subseteq dom(e.attTyp(a))$,
- für_alle $a \in e.A : \pi_1(e.card(a)) \leq |o.value(a)| \leq \pi_2(e.card(a))$

Das bedeutet, dass die Attributnamen des Objekts eine Obermenge der Attributnamen der Entität sind, dass die zugeordneten Werte Ausprägungen des korrekten Typs sind und dass die Anzahl der Attributwerte zwischen Minimal- und Maximalkardinalität der Entität liegt. □

Beispiel 3.7 (Objekt)
Das Objekt "Professor Roger Kaputnik" würde modelliert als Tupel

(A = { Name, Vorname, Fach, Telefonnummer},
value = { (Name, { "Kaputnik" }), (Vorname, { "Roger" }),
(Fach, { "Datenmanagement" }), (Telefonnummer, { "08154711" }) })

Man kann leicht überlegen, dass dieses Objekt Instanz der Entität Professor ist.

Definition 3.8 (mathematisches Modell für Beziehungen)
Eine Beziehung ist ein Tupel aus folgenden Teilen:

- eine Menge von Rollennamen $Rolle \subset ID$,
- eine Abbildung $rollenTyp : Rolle \to ID$ von Rollennamen in Entitätsnamen,
- eine Abbildung $card : Rolle \to \mathbb{N} \times \mathbb{N} \cup \{\infty\}$ von Rollennamen in Kardinalitäten

Die Menge aller Beziehungen heiße *Bzhg*. □

Beispiel 3.9 (Beziehung)
Wir betrachten eine ternäre Beziehung zwischen den Entitäten Person, Zeitpunkt und Raum "hält sich auf in". Person sei *-fach beteiligt, Zeitpunkt ebenfalls und Raum einfach. Dann ist die mathematische Repräsentation wie folgt:

(*Rolle* = { Teilnehmer, Termin, Ort },
rollenTyp = { (Teilnehmer, Person), (Termin, Zeitpunkt), (Ort, Raum) },
card = { (Teilnehmer, (0,∞)), (Termin, (0,∞)), (Ort, (1,1)) })

Anmerkung: Im jetzigen Modell haben Beziehungen keine Attribute. Man kann aber dann in die Beziehung eine weitere Entität mit Kardinalität 1 aufnehmen, die die Attribute der Beziehung trägt.

Definition 3.10 (Verbindung)
Eine *Verbindung von Objekten aus der Menge O* besteht aus folgenden Teilen:

- eine Menge *Rolle* von Rollennamen,
- eine Abbildung $value : Rolle \to O$ von Rollen in konkrete Objekte

□

Definition 3.11 (Verbindung als Instanz einer Beziehung)
Eine *Verbindung v ist Instanz einer Beziehung b* (i.Z. v **instanziiert** b), gdw. folgendes gilt:

- $v.Rolle = b.Rolle$,
- für_alle $r \in v.Rolle : v.value(r)$ **instanziiert** $b.rollenTyp(r)$

Das heißt, die Menge aller Rollen der Verbindung ist gleich der der Beziehung und die Typen der Objekte passen auf die der Beziehung. □

Beispiel 3.12 (Verbindung)
Die folgende Verbindung ist Instanz der Beziehung aus Beispiel 3.9. Auffällig ist, dass wirklich komplette Objekte als Werte der Funktion vorkommen.

```
( Rolle = { Teilnehmer, Termin, Ort },
  value = { (Teilnehmer,
              ( A     = { Name, Fach, Telefonnummer },
                value = {( Name, { "Kaputnik" } ), ( Vorname, { "Roger" } ),
                         (Telefonnummer, { 123456 } )})
            (Termin,
              ( A     = { Datum, Uhrzeit },
                value = {( Datum, { 1.3.2007 } ), ( Uhrzeit, { 7:45 } )})
            (Ort,
              ( A     = { Raumnummer },
                value = {( Raumnummer, { "Raum_1.1.07" } )})
          }
)
```

Anmerkung: Um die Notation zu vereinfachen, werden wir in Verbindungen statt kompletter Objekte nur die Abbildung für Schlüsselattribute aufführen.

Definition 3.13 (Verbindungsmenge paßt auf Beziehung)
Eine *Menge von Verbindungen $V = \{v_1, \ldots, v_n\}$ paßt auf Beziehung b* (i.Z. V **paßt_auf** b), gdw. folgendes gilt:

- für_alle $i : v_i$ **instanziiert** b,
- für_alle $r \in b.Rolle$ mit $andereRollen_{r,b} := b.Rolle \setminus \{r\}$,
  $P_r := \{ \; M \subseteq V \mid$
  $M = \{v_i, v_j \in V \mid$ für_alle $x \in andereRollen_{r,b} :$
  $v_i.value(x) = v_j.value(x)\}\} :$
  für_alle $M \in P_r : |M| \leq \pi_2(b.card(r))$

Das heißt: Alle Verbindungen sind Instanzen der Beziehung. Außerdem schaut man sich für jede Rolle r die Mengen von Verbindungen an, die außer in dieser Rolle in allen sonstigen Werten übereinstimmen. Diese Menge von Mengen P_r ist eine Partition von V. Jede Elementmenge darf höchstens so viele Elemente haben, wie die Maximalkardinalität vorgibt. □

Anmerkung: Die Prüfung der Untergrenze ist für eine Verbindungsmenge nicht möglich, weil sie nur existierende Verbindungen und deren Objekte enthält. Die aufgrund einer vorgegebenen Objektmenge möglichen Verbindungen sind unbekannt!

Beispiel 3.14 (Verbindungsmenge paßt auf Beziehung)
Wir betrachten obige ternäre Beziehung "hält sich auf in" zwischen Personen, Zeitpunkten und Räumen. Die Verbindungsmenge sei wie folgt (mit vereinfachten Objekten tabellarisch aufgeschrieben):

Verbindung	Teilnehmer	Termin	Ort
v_1	Paula	8:00	4711
v_2	Hugo	8:00	0815
v_3	Otto	8:00	0815
v_4	Otto	9:00	0815
v_5	Otto	9:00	4711

Die Verbindungsmenge $\{v_1, v_2, v_3, v_4\}$ paßt auf die Beziehung; das gilt aber nicht für $\{v_1, \ldots, v_5\}$, denn dann ist $P_{Ort} = \{\{v_1\}, \{v_2\}, \{v_3\}, \{v_4, v_5\}\}$ und das Element $\{v_4, v_5\}$ verletzt die Maximalkardinalität 1 für Rolle "Ort".

Definition 3.15 (E/R-Modell (mathematisch))
Ein E/R-Modell M enthält folgende Teile:

- eine Menge $EntID \subset ID$ von Entitätsnamen,
- eine Menge $E \subset Ent$ von Entitäten,
- eine Abbildung $ent : EntID \to E$ von Entitätsnamen auf die zugehörigen Entitäten,
- eine Menge $BezID \subset ID$ von Beziehungsnamen,
- eine Menge $B \subset Bzhg$ von Beziehungen,

- eine Abbildung $bzhg : BezID \to B$ von Beziehungsnamen auf die zugehörigen Beziehungen mit der Bedingung

$$\text{für\_alle } bName \in BezID, b := bzhg(bName), r \in b.Rolle : \\ b.rollenTyp(r) \in EntID$$

□

Wir kommen jetzt zu den konkreten Objekten und ihren Verbindungen:

Definition 3.16 (Objektnetz)
Ein *Objektnetz* besteht aus folgenden Teilen:

- eine Menge von Objekten $O := \{o_1, \ldots, o_n\}$
- eine Menge von Verbindungen $V := \{v_1, \ldots, v_m\}$ für O

□

Definition 3.17 (Abstraktionsfunktion)
Eine *Abstraktionsfunktion abstr* für ein Objektnetz N und ein E/R-Modell M ist eine Abbildung mit folgenden Eigenschaften:

- $abstr : (N.O \cup N.V) \to (M.EntID \cup M.BezID)$
- für_alle $o \in N.O : o$ **instanziiert** $M.ent(abstr(o))$
- für_alle $v \in N.V : v$ **instanziiert** $M.bzhg(abstr(v))$

Das heißt, *abstr* ist eine Funktion, die Objekte auf Entitäten und Verbindungen auf Beziehungen abbildet. Außerdem muß die Funktion so sein, dass die Objekte und Verbindungen Instanzen der Entitäten und Beziehungen sind (siehe Bild 3.3). □

Definition 3.18 (Instanz eines E/R-Modells (mathematisch))
Eine Objektnetz $N = (O, V)$ ist eine Instanz eines E/R-Modells M gdw. es eine Typfunktion *abstr* für N in M gibt mit folgenden Eigenschaften:

$$\text{für\_alle } bName \in M.BezID,\ b := M.bzhg(bName), \\ V_{bName} := \{v \in V : abstr(v) = bName\} :$$

- V_{bName} **paßt_auf** b

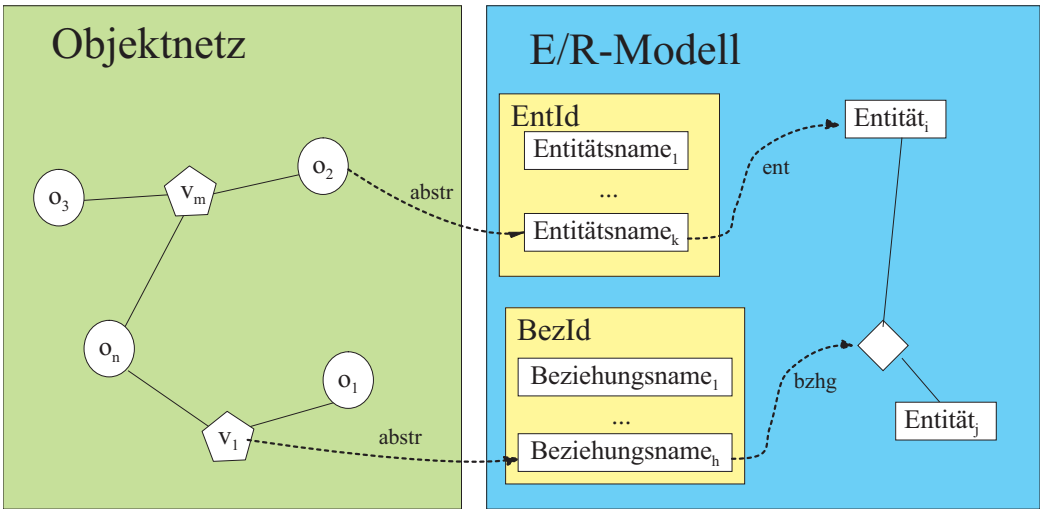


Abbildung 3.3: Abstraktion zwischen Objektnetz und E/R-Modell

- für_alle $r \in b.Rolle$:
  $andereRollen_{r,b} := b.Rolle \setminus \{r\}$,
  $F_{r,b} := \{f : andereRollen_{r,b} \to O \mid abstr(f(x)) = b.rollenTyp(x)\}$,
  $P_{r,b} := \bigcup_{f \in F_{r,b}} \{\{v \in V_{bName} \mid v \supseteq f\}\}$,
  für_alle $M \in P_{r,b} : |M| \geq \pi_1(card_b(r))$

Das heißt, die Menge der Verbindungen, die einer Beziehung b zugeordnet sind, paßt von der Kardinalität zu dieser Beziehung (Maximalkardinalität wird eingehalten).

Die zweite (zugegebenermaßen komplizierte ;-) Bedingung stellt sicher, dass für jede Rolle r einer Beziehung b für jede mögliche Belegung der Partnerrollen ($andereRollen_{r,b}$) durch konkrete Objekte (die Belegungskombinationen sind dargestellt durch $F_{r,b}$) mindestens so viele Verbindungen existieren, wie die Minimalkardinalität bei Rolle r verlangt. □

Kapitel 4

Normalisierung

Die bisherigen Ausführungen zur Modellierung haben zwar das Instrumentarium für eine Modellierung bereitgestellt, beinhalten jedoch keine qualitativen, wertenden Aussagen über das entstehende Modell. Die einzige Restriktion für ein Modell, die sich daraus direkt ableiten lässt ist, dass die Syntax und Semantik des Metamodells eingehalten werden müssen.

Inhalt dieses Kapitels sind nun Ansätze zur Bewertung eines Modells und dessen Qualitätsverbesserung.

4.1 Qualitätsmerkmale eines Modells

Ein Modell ist dann gut, wenn es

- die *Anwendungsdomäne* treffend und vollständig widergibt,
- keine über die Anwendungsdomäne hinausgehenden Daten enthält,
- durch Integritätsbedingungen Widerspruchsfreiheit gewährleistet ,
- und die optimale *Nutzung* der Daten sicherstellt.

Während die Nähe des Modells zur Anwendungsdomäne sehr speziell, wenig systematisierbar und in der Praxis von den kommunikativen und Abstraktions- Fähigkeiten des Modellierers und der beteiligten Fachbereiche abhängig ist, gibt es für die Nutzbarkeit des Modells schematische und damit gut prüfbare Kriterien.

4.1.1 Atomare Daten

Um sicherzustellen, dass für jede Operation der dazu benötige Eingriff so klein wie möglich gehalten wird, müssen Daten stets in atomarer Form abgelegt werden.

- Jedes *Attribut* sollte einen kleinstmöglichen wesentlichen Informationsbaustein repräsentieren. (Beispiel: Wenn in der Anwendung Name und Vorname jeweils wichtig sein können, muss es dafür getrennte Attribute geben.)
- Jede *Beziehung* sollte die direkte Verbindung zwischen zwei Entitäten beschreiben, also die kürzest mögliche. (Beispiel: Zwischen je zwei Personen gibt es eine "ist_Elternteil_von"-Beziehung, nicht eine "ist_Großelternteil_von"-Beziehung.)
- Schließlich sind auch *Integritätsbedingungen* auf den kleinsten möglichen Kontext zu reduzieren bzw. in mehrere Teile zu zerlegen.

4.1.2 Redundanzfreiheit

Ein zentrales Ziel der Modellierung ist, ein Modell zu erhalten, das redundanzfrei ist bezüglich

- *Daten*, d.h. eine Information soll in einem Modell nur einmal, nicht mehrfach gespeichert sein,
- *Beziehungen*, d.h. auch Beziehungen zwischen je zwei Objekten sollen genau einmal gespeichert werden,
- *Typen*, d.h. Informationen desselben Art sollen auch mit demselben Typ charakterisiert werden.

Wenn diese Prinzipien im Modell konsequent berücksichtigt werden, führt dies dazu, dass für jedes Datum ein eindeutiger Ort zu seiner Speicherung definiert ist.

Der Vorteil dieses Vorgehens ist zunächst die offensichtliche Einsparung an Speicherplatz. Wichtiger ist jedoch, dass darüber hinaus sichergestellt ist, dass Änderungen eines Datums jeweils nur genau an einer Stelle erfolgen.

Damit werden Änderungsoperationen so simpel wie möglich und sogenannte *Anomalien* bei der Behandlung von Daten vermieden.

Definition 4.1 (Anomalien)
Es gibt folgende Arten von Anomalien:

- *Änderungsanomalie:* Die Änderung eines einzigen Datums in der realen Welt führt zu mehreren Änderungen im Modell.
- *Einfügeanomalie:* Eine vorhandene Information kann im Modell nicht hinterlegt werden, weil sie als abhängige Information erst nach Eintreten eines bestimmten Ereignisses im Modell Niederschlag findet.
- *Löschanomalie:* In Umkehrung der Einfügeanomalie wird mit dem Löschen einer Information eine andere Information ebenfalls gelöscht, die für sich jedoch weiter Bestand haben müsste.

□

Beispiel 4.2 (Anomalien)
Das Informationsmodell eines Verlages sieht die Entität Buchbestand vor, mit den Attributen (Titel, Erscheinungsjahr, ISBN, Autorenname, Autorenanschrift).

Nehmen wir nun an, in diesem Verlag seien von ein und demselben Autor "Paulo Coelho" außer "Der Alchimist" auch "Der Zahir" und "Elf Minuten" erschienen und damit im Buchbestand verfügbar. Dann gäbe es drei Einträge in der Entität Buchbestand, in denen der Autorenname "Paulo Coelho" und dazu seine Anschrift gespeichert wären.

Beispiel für das Auftreten von Anomalien
Buchbestand:

Titel	Jahr	ISBN-Nr	Autorenname	Autorenanschrift
Der Alchimist	1988	3257061269	Paulo Coelho	80808 München, Schlossallee 4711
Der Zahir	2006	3257235801	Paulo Coelho	80808 München, Schlossallee 4711
Elf Minuten	2003	3257063733	Paulo Coelho	80808 München, Schlossallee 4711
...	...	...	...	...

- Würde die Adresse sich ändern, müsste in allen drei Einträgen geändert werden (→ Änderungsanomalie).
- Vor Erscheinen des ersten Buches hätte der Verlag mit diesem Datenmodell keine Möglichkeit gehabt, die Anschrift des Autors in seine Daten aufzunehmen, was unter Umständen der Aufnahme von Geschäftskontakten und damit einem ersten Erscheinen eines Buches im Wege gestanden hätte (→ Einfügeanomalie).

- Umgekehrt würde die Anschrift gelöscht werden, wenn der Verlag die drei Werke aus seinem Angebot nehmen würde, obwohl vielleicht die Veröffentlichung eines neuen Werkes anstünde und damit die Anschrift wieder gebraucht würde (→ Löschanomalie).

Anmerkung: Zwischen den beiden Postulaten, Daten atomar und redundanzfrei zu halten, besteht insofern ein Zusammenhang, als das Erkennen und damit Vermeiden der Datenredundanz immens vereinfacht wird, wenn Informationen feingranular vorliegen.

Beispiel: Wenn dem Kunden die Attribute Name, Anschrift und Zustellgebiet zugeordnet sind, wobei das Zustellgebiet durch die Postleitzahl definiert wird, erkennt man nicht sofort, dass dieses Zustellgebiet bereits Teil der Anschrift ist. Das sieht anders aus, wenn die Anschrift in ihren Bestandteilen Straße, Hausnummer, PLZ und Ort dargestellt ist.

4.2 Normalformen

In diesem Abschnitt wird eine Methode zur Überprüfung eines Modelles bzgl. der eben aufgeführten Qualitätsziele vorgestellt. Gleichzeitig kann diese Methode genutzt werden, um ein beliebiges Mdeoll schrittweise in ein Modell, das diesen Qualitätskriterien genügt, zu überführen (→ Normalisierung).

Die Normalisierung zielt darauf ab, dass

- alle Informationen atomar vorliegen,
- in den Entitäten nur jeweils zusammengehörende Informationen liegen
- und keine Redundanzen bestehen.

4.2.1 Erste Normalform (1NF)

Definition 4.3 (1NF)
Ein *Modell ist in erster Normalform*, wenn jedes Attribut jeder Entität in atomarer Form vorliegt. □

Das bedeutet, dass in einem Modell, das in der ersten Normalform ist, kein mehrwertiges Attribut vorkommen kann.

Beispiel: In der oben genannten Entität “Buchbestand” ist die erste Normalform bereits durch das nicht-atomare Attribut “Anschrift” verletzt. Ein weiterer Verstoß gegen die 1NF würde eintreten, sobald ein Buch mit mehreren Autoren erscheinen würde. In dem Fall müssten die Attribute “Autorenname” und “Autorenanschrift” mehrfach belegt werden, was nur durch einen nicht atomaren Datentyp (array) möglich wäre.

Um die Verstöße gegen die 1NF aus dem Modell zu entfernen, müsste die Entität anders dargestellt werden, bspw. wie im Folgenden. Zur Verdeutlichung werden die beiden kritischen Attribute in zwei Schritten geändert. Zuerst wird die Anschrift in atomare Bestandteile zerlegt, was für die praktische Nutzung sinnvoller sind. Als zweites wird das Problem möglicher Autorenteams gelöst.

Beispiel 4.4 (Herstellen der 1NF)

1. Herstellen der 1NF im Attribut Autorenanschrift

   Buchbestand:

Titel	Jahr	ISBN-Nr	Autorenname	PLZ	Ort	Straße
Der Alchimist	1988	3257061269	Paulo Coelho	80808	München	Schlossallee 4711
Der Zahir	2006	3257235801	Paulo Coelho	80808	München	Schlossallee 4711
Elf Minuten	2003	3257063733	Paulo Coelho	80808	München	Schlossallee 4711
...	...	...	...	...	...	...

2. Herstellen der 1NF im Attribut Autor
   Da Coelho seine Bücher in der Regel allein schreibt, wird der Entität ein weiterer Eintrag hinzugefügt, und zwar ein Buch, das von einem Autorenteam Kopietz/Sommer geschrieben wurde.

   Buchbestand:

Titel	Jahr	ISBN-Nr	Autorenname	PLZ	...
Der Alchimist	1988	3257061269	Paulo Coelho	80808	...
Der Zahir	2006	3257235801	Paulo Coelho	80808	...
Elf Minuten	2003	3257063733	Paulo Coelho	80808	...
R.U.D.I. der Held	2003	3831503443	Gerit Kopietz, Jörg Sommer	10117	...
...	...	...	...	...	...

   Dieser Eintrag verletzt die 1NF, da zwei Autoten unter Autorname aufgeführt sind, die aber jeder für sich erkennbar sein müssen. Für die beiden ist auch zunächst nur eine Anschrift vorgesehen, was in der Regel nicht ausreicht. Um diesen Konflikt aufzulösen, gibt es in der vorhandenen Struktur nur die Möglichkeit, das Buch mehrfach zu führen.

Buchbestand:

Titel	Jahr	ISBN-Nr	Autorenname	PLZ	Ort	Straße
Der Alchimist	1988	3257061269	Paulo Coelho	80808	München	Schlossallee 4711
Der Zahir	2006	3257235801	Paulo Coelho	80808	München	Schlossallee 4711
Elf Minuten	2003	3257063733	Paulo Coelho	80808	München	Schlossallee 4711
R.U.D.I. der Held	2003	3831503443	Gerit Kopietz	10117	Berlin	Parkstraße 1
R.U.D.I. der Held	2003	3831503443	Jörg Sommer	10117	Berlin	Parkstraße 3
...	...	...	...	...	...	...

Diese Umnutzung der Buchbestand-Struktur hat zur Folge, dass der Schlüssel erweitert werden muss. Während ursprünglich die ISBN-Nummer zur Identifikation eines Eintrags ausreichend war, müssen wir nun bspw. den Autorennamen dazunehmen, um wieder einen gültigen Schlüssel zu erhalten.

4.2.2 Zweite Normalform (2NF)

Definition 4.5 (funktionale Abhängigkeit)
Eine *Attributmenge B ist funktional abhängig von einer Attributmenge A* (i.Z. $A \rightsquigarrow B$), gdw. aus der Übereinstimmung zweier Objekte o_1 und o_2 in den Attributwerten in A auch die Übereinstimmung in allen Werten für Attribute aus B folgt. □

Anmerkung: Funktionale Abhängigkeit ist eine strukturelle Eigenschaft einer Relation. Man kann sie nicht einfach aus einer Objektmenge ableiten (dort können zufälligerweise bestimmte Übereinstimmungen auftreten), sondern man muss sie aus dem Wissen über die Anwendungsdomänen gewinnen.

Definition 4.6 (transitive Hülle)
Die transitive Hülle einer Relation $\rho \subseteq X \times X$ wird mit ρ^+ bezeichnet und ist definiert als

$$\rho^+ := \{(x, z) \in X \times X \mid x\rho z \vee (\text{gibt\_es } y \in X : x\rho y \wedge y\rho^+ z)\}$$

□

Definition 4.7 (2NF)
Ein *Modell ist zweiter Normalform*, wenn es in erster Normalform ist und jedes Attribut einer jeden Entität vom gesamten Schlüssel der Entität funktional abhängig ist. □

Das bedeutet, dass in einem Modell, das in der zweiten Normalform ist, keine Entität existieren darf, in der es ein Attribut gibt, das nur von einem Teil des Schlüssels abhängt.

Beispiel 4.8 (Herstellen der 2NF)
In der eben erweiterten Entität "Buchbestand" mussten wir, um den Fall mehrerer Autoren in der Struktur unterzubringen, den Schlüssel erweitern auf ISBN-Nummer und Autorenname. Trotzdem hängen aber weiterhin Attribute wie z.B. der Buchtitel allein von der ISBN-Nummer ab.

Um zu erreichen, dass jedes Attribut vom gesamten Schlüssel abhängt, muss die Tabelle zerlegt werden, in einen Teil mit Attributen zum Autor (funktional abhängig vom Autorennamen) und einen zweiten Teil mit Attributen zum Buchbestand (funktional abhängig von ISBN-Nummer und Autor zusammen oder sogar nur ISBN-Nr).

Buchbestand:

Titel	Jahr	ISBN-Nr	Autorenname
Der Alchimist	1988	3257061269	Paulo Coelho
Der Zahir	2006	3257235801	Paulo Coelho
Elf Minuten	2003	3257063733	Paulo Coelho
R.U.D.I. der Held	2003	3831503443	Gerit Kopietz
R.U.D.I. der Held	2003	3831503443	Jörg Sommer
...	...	...	...

Buchautor:

Autorenname	PLZ	Ort	Straße
Paulo Coelho	80808	München	Schlossallee 4711
Gerit Kopietz	10117	Berlin	Parkstraße 1
Jörg Sommer	10117	Berlin	Parkstraße 3
...	...	...	...

4.2.3 Dritte Normalform (3NF)

Definition 4.9 (3NF)
Ein *Modell ist in dritter Normalform*, wenn es in erster Normalform ist und jedes Nicht-Schlüssel-Attribut einer jeden Entität von jedem Schlüssel der Entität direkt funktional abhängig ist. □

Das bedeutet, dass es keine transistive funktionale Abhängigkeit zwischen Nicht-Schlüsselattributen geben darf, d.h. für jedes Nicht-Schlüsselattribut gilt, dass es sich nicht mittelbar aus einem oder mehreren anderen Attribut(en) ergeben darf.

Beispiel 4.10 (Herstellen der 3NF)
In unserem Beispiel ergibt sich der Ort aus der PLZ, unabhängig vom Autor. Die Zuordnung des Ortes zum Autor ist also indirekt aus der PLZ des Autors ableitbar. Um diese transitive Abhängigkeit aus der Tabelle zu entfernen, könnte man bspw. die Zuschlüsselung des Ortes zu einer PLZ aus der

Autorentabelle entfernen und bekäme dann folgendes Modell:

Autorenschaft:

ISBN-Nr	Autorenname
3257061269	Paulo Coelho
3257235801	Paulo Coelho
3257063733	Paulo Coelho
3831503443	Gerit Kopietz
3831503443	Jörg Sommer
...	...

Buch:

Titel	Jahr	ISBN-Nr
Der Alchimist	1988	3257061269
Der Zahir	2006	3257235801
Elf Minuten	2003	3257063733
R.U.D.I. der Held	2003	3831503443
R.U.D.I. der Held	2003	3831503443
...	...	...

Ort:

PLZ	Ort
80808	München
10117	Berlin
...	...

Buchautor:

Autorenname	PLZ	Straße
Paulo Coelho	80808	Schlossallee 4711
Gerit Kopietz	10117	Parkstraße 1
Jörg Sommer	10117	Parkstraße 3
...	...	...

Auch die transitiven Abhängigkeiten bilden Redundanzen im Modell, wie man an dem Beispiel erkennen kann: die Information darüber, dass die PLZ 10117 immer Berlin impliziert, die 80808 dagegen München, wird solange nicht explizit, wie nicht eine eigene Entität zu PLZ-Zuordnung eingeführt ist.

4.2.4 Boyce-Codd-Normalform (BCNF)

Definition 4.11 (Boyce-Codd-NF)
Ein *Modell ist in Boyce-Codd-Normalform*, wenn sie in erster Normalform ist und jedes Attribut einer jeden Entität vom Schlüssel der Entität direkt funktional abhängig ist. □

Das bedeutet, dass es keine transistive Abhängigkeit zwischen Attributen geben darf, d.h. es darf kein Attribut existieren, das sich mittelbar aus einem oder mehreren anderen Attribut(en) ergibt. Die Boyce-Codd-Normalform ist also eine Verschärfung der 3NF, da dieselbe Bedingung gestellt wird, allerdings außer für Nicht-Schlüsselattribute auch für Schlüsselattribute.

Ein Beispiel dafür liefert die genauere Betrachtung der Postleitzahlenlogik.

Beispiel 4.12 (Herstellen der Boyce-Codd-Normalform)
In unserem Beispiel wurde vereinfachend jeder PLZ ein Ort zugeordnet. Die Entität "Ort" wurde mit dem Schlüssel PLZ versehen. Tatsächlich gibt es aber eine Zuordnungsvorschrift, die jeder Adresse eine PLZ zuordnet. (Diese Entität ist im Folgenden "Rolf" genannt, analog zur ersten Ausgabe des Postleitzahlenbuchs bei Einführung der fünfstelligen PLZ.)

Rolf:

Ort	Straße	Hausnummer	PLZ
München	Schlossallee	4711	80808
Berlin	Parkstraße	1	10117
Berlin	Parkstraße	3	10117
...	...	...	...

In dieser Entität kommen sowohl {Ort, Straße, Hausnummer} als auch {PLZ, Straße, Hausnummer} als Schlüssel in Frage. Damit sind alle vorhandenen Attribute auch Schlüsselattribute, so dass für die Einhaltung der 3NF nicht nach transitiven Abhängigkeiten gesucht werden muss, anders bei der Untersuchung bzgl. der BCNF. Hier zeigt sich, dass die PLZ funktional von den drei Schlüsselattributen Ort, Straße und Hausnummer abhängt. Gleichzeitig gilt aber auch die bereits verwendete Abhängigkeit, dass von der PLZ auf den Ort geschlossen werden kann.

Leider ergibt die Auflösung dieser Abhängigkeit ein unbefriedigendes Ergebnis, was meist bezeichnend für die praktische Einsetzbarkeit der BCNF ist. Daher verwendet man bei der ER-Modellierung oft nur die 3NF:

Straßenname:

Ort	Straße	Straßencode
München	Schlossallee	1
Berlin	Parkstraße	2
Dorfen	Hauptstraße	3
...	...	...

Rolf:

Straßencode	Hausnummer	PLZ
1	4711	80808
2	1	10117
...	...	...

In diesen beiden Entitäten findet sich zwar keine transitive Abhängigkeit mehr, auch nicht unter Schlüsselattributen, aber das entstandene Modell wirkt gekünstelt.

Kapitel 5

Vorgehen

5.1 Verfahren zur Erstellung von E/R-Modellen

Neben der Methode der E/R-Modellierung selbst gibt es auch Ideen zum Vorgehen bei der Modellierung. Diese haben immer zum Ziel, nicht nur irgendein Modell zu erstellen, sondern auf möglichst direktem Weg zu einem Modell zu kommen, das die oben eingeführten Qualitätsmerkmale erfüllt.

5.1.1 Dekomposition

Während die im letzten Kapitel beschriebenen Normalformen jeweils die Redundanzfreiheit innerhalb einer Entität untersuchen, ist das Ziel einer jeden Modellbildung auch die Redundanzfreiheit zwischen verschiedenen Entitäten sicherzustellen.

Um dies zu erreichen, liegt es nahe, zunächst mit sehr großen Entitäten zu beginnen, in denen alle relevanten Attribute des Betrachtungsraumes gesammelt sind. So ist sichergestellt, dass es zumindest zu Beginn der Modellierung keine Redundanzen zwischen Entitäten gibt. Eine solche Megaentität wird zwar zunächst wenig Bezug zur Anwendungsdomäne erkennen lassen und in ihrer Breite nicht sinnvoll nutzbar sein, kann aber durch geeignete Transformationen in mehrere Entitäten mit gleichem Informationsgehalt überführt werden.

Nach diesem Prinzip funktionieren *Dekompositionverfahren*. Dekomposition ist die älteste bekannte Entwurfstechnik für die Modellierung relationaler

Datenbanken.

Wenn man nun eine geeignete Vorgehensweise findet, die sicherstellt, dass bei jedem Zerlegungsschritt Redundanzen einer Entität aufgelöst werden, ohne Redundanzen zwischen Entitäten zu schaffen, kann man in gesicherten Schritten zu einem guten Modell kommen.

5.1.2 Regeln für semantikerhaltende Umformungen

Bei der Zerlegung von Entitäten ist darauf zu achten, dass das Modell nicht plötzlich Objektnetze als Instanzen zuläßt, die vorher nicht möglich waren.

Um das sicherzustellen, müssen bei der Dekomposition bestimmte Regeln eingehalten werden.

Definition 5.1 (Abhängigkeitstreue)
Für eine Attributmenge und darauf gegebenen funktionalen Abhängigkeiten heißt ein Modell mit entsprechenden Entitäten und nur mit einfachen Fremdschlüsselbeziehungen (ein "Relationenschema") *abhängigkeitstreu*, wenn die im Modell vorhandenen Schlüsselabhängigkeiten äquivalent zu den gegebenen funktionalen Abhängigkeiten sind. □

Beispiel 5.2 (Abhängigkeitstreue)
Gegeben seien die Attribute $A = \{PLZ, Ort, Strasse, Hausnummer\}$ mit

$$FD_A = \{ \begin{array}{l} \{Ort, Strasse, Hausnummer\} \rightsquigarrow \{PLZ\}, \\ \{PLZ\} \rightsquigarrow \{Ort\}\} \end{array}$$

Sei Modell M mit einer Entität E gegeben mit $E.A = A$, $E.S = \{Ort, Strasse, Hausnummer\}$, dann ist die Schlüsselabhängigkeit in E

$$\{\{Ort, Strasse, Hausnummer\} \rightsquigarrow \{Ort, Strasse, Hausnummer, PLZ\}\}$$

nicht äquivalent zu FD_A und damit M nicht abhängigkeitstreu.

Definition 5.3 (Verbundtreue)
Eine Zerlegung einer Entität in mehrere andere durch Zerlegung in sich überlappende Attributmengen ist *verbundtreu* (oder *verlustfrei*), wenn durch Kreuzproduktbildung (join) entlang gleicher Attribute mit gleichen Werten keine falschen Objekttupel entstehen. □

Beispiel 5.4 (Verbundtreue)
Sei $E.A = \{X, Y, Z\}$, $E_1.A = \{X, Y\}$ und $E_2.A = \{Y, Z\}$.

Für $FD = \{X \rightsquigarrow Y, Y \rightsquigarrow Z\}$ ist diese Aufteilung von E in E_1 und E_2 verbundtreu, für $FD = \{X \rightsquigarrow Y, Z \rightsquigarrow Y\}$ ist sie es nicht.

gut

X	Y	Z
1	2	3
4	2	3

$\overset{Zerlegung}{\Longrightarrow}$

X	Y
1	2
4	2

Y	Z
2	3

$\overset{join}{\Longrightarrow}$

X	Y	Z
1	2	3
4	2	3

schlecht

X	Y	Z
1	2	3
4	2	5

$\overset{Zerlegung}{\Longrightarrow}$

X	Y
1	2
4	2

Y	Z
2	3
2	5

$\overset{join}{\Longrightarrow}$

X	Y	Z
1	2	3
4	2	5
1	2	5
4	2	3

Satz 5.5 (Verbundtreue)
Eine Aufteilung von Entität E in E_1 und E_2 ist verbundtreu bezüglich funktionalen Abhängigkeiten FD, wenn

$$\text{gibt\_es } i \in \{1, 2\} : ((E_1.A \cap E_2.A) \rightsquigarrow E_i.A) \in FD^+$$

. □

5.1.3 Normalisierung

Das Verfahren der *Normalisierung* ist ein Dekompositionsverfahren, das aus den Normalform-Regeln Transformationsschritte ableitet. In der reinen Form würde man mit einer einzigen Entität beginnen, in der alle Attribute der Anwendungsdomäne gesammelt sind. Diese Entität kann man Schritt für Schritt in die Normalformen bringen.

Folgende Dekompositionsschritte werden bei der Normalisierung durchgeführt:

Herstellen der 1NF , d.h.

- Darstellen aller Attribute in *atomarer* Form, d.h. dass alle Attribute der Entität so lange in verfeinerte Attribute aufgeteilt werden müssen, bis keine in der fachlichen Domäne sinnvolle Aufteilung mehr möglich ist.

- Auflösen von *Wiederholungsgruppen*, also Überführung von mehrwertigen Attributen in mehrere Einträge mit jeweils einfacher Attributbelegung.
- Festlegen von *Schlüsselattributen*, d.h. alle möglichen Tupel von Attributen, die identifizierend sind, müssen benannt werden.

Herstellen der 2NF , d.h.

- Identifikation *funktionaler Abhängigkeiten* von Nicht-Schlüssel-Attributen zu Teilen von Schlüsselattributen.
- *Auslagern* derjenigen Attribute, die von nur einem Teil der Schlüsselattribute abhängig sind, aus der betroffenen Entität bis zum Erreichen der 2NF.
- Bildung *Neuer Entitäten* aus den augelagerten Attributen und der Untermenge von Schlüsselattributen, von denen diese abhängig sind.

Herstellen der 3NF , d.h.

- Identifikation *transitiver Abhängigkeiten* von Nicht-Schlüssel-Attributen zu Teilen von Schlüsselattributen.
- *Auslagern* der unerlaubten Attribute aus der betroffenen Entität bis zum Erreichen der 3NF.
- Bildung *Neuer Entitäten* aus den augelagerten Attributen und den Schlüsselattributen, von denen diese abhängig sind.

Da die Überführung in die BCNF in der Praxis keine große Bedeutung hat, und da der Schritt analog dem zuletzt aufgführten wäre, sei hier darauf verzichtet.

Beispiel 5.6 (Fallbeispiel zur Dekomposition)
Gegeben sei die Entität für einen Projektmitarbeiter (siehe Bild 5.1).

Zur Demonstration sei eine Objektmenge gegeben als

Projektmitarbeiter:

NameUnd-Vorname	Ausweis-Nr	Proj-Nr	Einsatz PLZ	Einsatzort	Datum von	Datum bis	Fach-gebiet
Klein Fred	1234	P1 P2	80851 20095	München Hamburg	1.2.07 1.7.07	31.7.07 31.8.07	DBMS UML
Gross Mike	4711	P2	20095	Hamburg	1.7.07	31.8.07	DB2 EAI
Hoch Lina	0815	P3	81377	München	1.6.07	31.10.07	SAP

Wir wenden das obige Verfahren an:

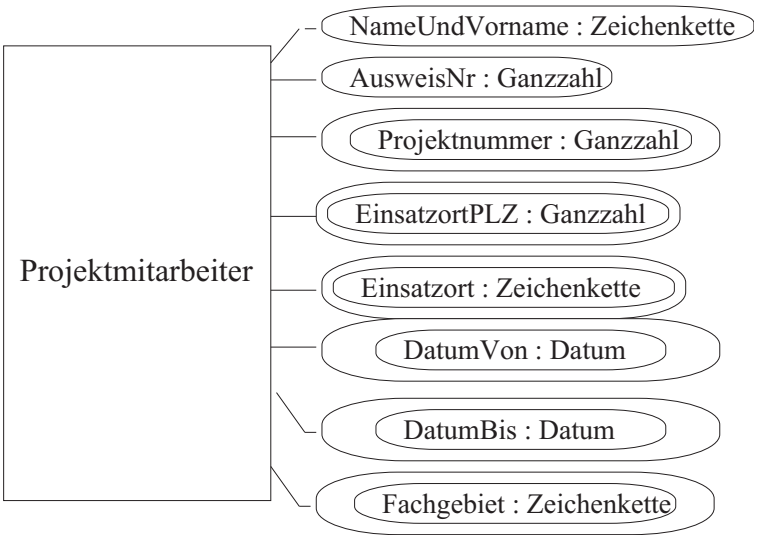


Abbildung 5.1: Beispiel-Basisentität für Dekomposition

Herstellen der 1NF , d.h.

- Darstellen aller Attribute in atomarer Form: Zerlegt werden sollte"NameUndVorname".
- Auflösen von Wiederholungsgruppen: Es ist nicht sauber modelliert, welche Wiederholungsgruppen wirklich zusammengehören. Nachdenken (!) ergibt, dass die Wiederholungsgruppen wie folgt aufgelöst werden:
  Projektmitarbeiter:

Name	Vor-name	Ausweis-Nr	Proj-Nr	Einsatz PLZ	Einsatzort	Datum von	Datum bis	Fach-gebiet
Klein	Fred	1234	P1	80851	München	1.2.07	31.7.07	DBMS
Klein	Fred	1234	P2	20095	Hamburg	1.7.07	31.8.07	UML
Gross	Mike	4711	P2	20095	Hamburg	1.7.07	31.8.07	DB2
Gross	Mike	4711	P2	20095	Hamburg	1.7.07	31.8.07	EAI
Hoch	Lina	0815	P3	81377	München	1.6.07	31.10.07	SAP

- Festlegen von Schlüsselattributen: In obiger Tabelle sind "AusweisNr", "ProjNr" und "Fachgebiet" Schlüssel.

Herstellen der 2NF , d.h.

- Identifikation funktionaler Abhängigkeiten:

$$FD = \{ \quad \{AusweisNr\} \rightsquigarrow \{Name, Vorname\},$$

$$\{AusweisNr, ProjNr\} \rightsquigarrow \{DatumVon, DatumBis\},$$
$$\{ProjNr\} \rightsquigarrow \{EinsatzPLZ, Einsatzort\}\}$$

- Auslagern der Attribute mit Teilabhängigkeiten und Bildung neuer Entitäten (diese Zerlegungen sind verbundtreu!)

Person:

AusweisNr	Name	Vorname
1234	Klein	Fred
4711	Gross	Mike
0815	Hoch	Lina

Projekt:

ProjNr	EinsatzPLZ	Einsatzort
P1	80851	München
P2	20095	Hamburg
P3	81377	München

Mitarbeit:

AusweisNr	ProjNr	DatumVon	DatumBis
1234	P1	1.2.07	31.7.07
1234	P2	1.7.07	31.8.07
4711	P2	1.7.07	31.8.07
0815	P3	1.6.07	31.10.07

Expertise:

AusweisNr	ProjNr	Fachgebiet
1234	P1	DBMS
1234	P2	UML
4711	P2	DB2
4711	P2	EAI
0815	P3	SAP

Herstellen der 3NF , d.h.

- Identifikation transitiver Abhängigkeiten: Es gibt eine funktionale Abhängigkeit in "Projekt"

$$FD = \{\{EinsatzPLZ\} \rightsquigarrow \{Einsatzort\}\}$$

  und damit eine transitive Abhängigkeit!

- Auslagern der Attribute mit trans. Abhängigkeiten und Bildung neuer Entitäten: "Projekt" wird zerlegt in "Projekt-neu" und "Ort".

Projekt-neu:

ProjNr	EinsatzPLZ
P1	80851
P2	20095
P3	81377

Ort:

EinsatzPLZ	Einsatzort
80851	München
20095	Hamburg
81377	München

5.1.4 Synthese

Während man bei der Dekomposition im Idealfall von einer einzigen Entität ausgehend durch sukzessive Zerlegung zu einem guten Modell gelangt, ist es auch möglich, mit einem Modell zu beginnen, das bereits aus einer Vielzahl von Entitäten besteht.

Da in der Analyse einer Anwendungsdomäme immer auch Zusammenfassungen von Attributen auftreten, die eine eigene fachliche Bedeutung haben, ist

es auch durchaus sinnvoll, diese Information mit aufzunehmen und in die Modellierung einzubeziehen. Damit kommt man zu einem Erstentwurf eines Modelles, das bereits einen soliden Eindruck macht. Allerdings kann man nicht davon ausgehen, dass damit automatisch auch die Entwurfskriterien erfüllt sind.

Beispiel 5.7 (Fallbeispiel zur Synthese)
In Interviews mit dem Fachbereich wurden folgende Entitäten identifiziert:

Person:

AusweisNr	Name	Vorname
1234	Klein	Fred
4711	Gross	Mike
0815	Hoch	Lina

Projekt:

ProjNr	EinsatzPLZ	Einsatzort
P1	80851	München
P2	20095	Hamburg
P3	81377	München

Projektplan:

ProjNr	DatumVon	DatumBis	Skills
P1	1.2.07	31.7.07	DBMS
P2	1.7.07	31.8.07	UML,DB2,EAI
P3	1.6.07	31.10.07	SAP

Hier fällt auf, dass die Entitäten Projekt und Projektplan den gleichen Schlüssel haben. Außerdem stehen die beiden Entitäten in einer 1:1 Beziehung zueinander. Tatsächlich können sie in eine einzige Entität zusammengefasst werden:

Projekt:

ProjNr	EinsatzPLZ	Einsatzort	DatumVon	DatumBis	Skills
P1	80851	München	1.2.07	31.7.07	DBMS
P2	20095	Hamburg	1.7.07	31.8.07	UML,DB2,EAI
P3	81377	München	1.6.07	31.10.07	SAP

Das mehrfache Attribut Skills ist damit allerdings nach wie vor vorhanden.

Es muss also durch geeignete Prüfung sichergestellt werden, dass Redundanzen zwischen den Entitäten erkannt und beseitigt werden, ebenso wie Redundanzen innerhalb einzelner Entitäten.

Man wird daher einerseits die bekannten Normalisierungsregeln anwenden, um funktionale Abhängigkeiten innerhalb einzelner Entitäten so weit als möglich in Schlüsselabhängigkeiten zu überführen. Dies führt zu einer weiteren Zerteilung des Modells. Anders als beim Dekompositionsansatz hat man nun aber eine neue Unsicherheit darüber, ob auch wirklich alle zusammengehörende Information in einer einzigen Entität zusammengefasst ist. Insbesondere kann durch das Normalisierungsverfahren keine Abhängigkeit zwischen Entitäten erkannt werden, da dieser Aspekt nicht in den Regeln der Normalform anthalten ist. Die Zusammenfassung zusammengehörender In-

formation in einer einzigen Entität ist also ein wesentliches Qualitätsmerkmal und Voraussetzung zur Erkennung von Querabhängigkeiten.

Daraus entsteht die Notwendigkeit, ein Modell durch Zusammenfassen von Entitäten so zu transformieren, dass alle gewünschten Qualitätsmerkmale erfüllt sind. Neben den bereits bekannten Regeln zum Semantikerhalt wird dabei die Minimalität der Entitäten angestrebt.

Wichtigster Indikator für das Erkennen zusammengehöriger Attribute sind die Beziehungen zwischen zwei Entitäten. Zwei Entitäten, die in einer 1:1-Beziehung zueinander stehen, können prinzipiell zusammengefasst werden.

5.1.5 Semantikerhalt beim Zusammenfassen von Entitäten

Ebenso wie bei der Zerlegung einer Entität muss das Ergebnis der Zusammenfassung zweier Entitäten dem ursprünglichen Modell inhaltlich gleichwertig sein. Es darf also keine Information verlorengehen, genauer gesagt darf das neu gewonnene Modell nicht weniger Ausagen über funktionale Abhängigkeiten zulassen als das ursprümgliche. Umgekehrt dürfen auch keine Objekte möglich werden, die zuvor nicht zum Modell gepasst hätten. Auch dazu geben die bereits bekannten Prüfkriterien von Modellen Auskunft.

Wir gehen im Folgenden davon aus, dass jede einzelne Entität eines vorliegenden Modells bereits normalisiert worden ist.

Definition 5.8 (Abhängigkeitstreue/Wiederholung)
Für eine Attributmenge und darauf gegebenen funktionalen Abhängigkeiten heißt ein Modell mit entsprechenden Entitäten und nur mit einfachen Fremdschlüsselbeziehungen (ein "Relationenschema") *abhängigkeitstreu*, wenn die im Modell vorhandenen Schlüsselabhängigkeiten äquivalent zu den gegebenen funktionalen Abhängigkeiten sind. □

Will man also bezüglich der Abhängigkeitstreue durch die Synthese zweier Entitäten keinen "Qualitätsverlust" erleiden, so dürfen dabei keine neuen funktionalen Abhängigkeiten, außer solchen zu Schlüsselattributen entstehen. Dies kann bspw. damit sichergestellt werden, dass

- die beiden Entitäten in einer *1:1* Beziehung zueinander stehen oder - falls die Entitäten bereits in normalisierter Form vorliegen -
- jedes abhängige Attribut der beiden Entitäten von jedem *Schlüssel* der beiden Entitäten funktional abhängig ist.

Beispiel 5.9 (Fallbeispiel zur Synthese)
Variation des obigen Beispiels ergibt folgende Entitäten:

Person:

AusweisNr	Name	Vorname
1234	Klein	Fred
4711	Gross	Mike
0815	Hoch	Lina

Projekt:

ProjNr	EinsatzPLZ	Einsatzort
P1	80851	München
P2	20095	Hamburg
P3	81377	München

Projektplan:

ProjName	DatumVon	DatumBis
BMW-MaterialDB	1.2.07	31.7.07
Schiffsverleih	1.7.07	31.8.07
BayernLB-DWH	1.6.07	31.10.07

Die Entitäten Projekt und Projektplan stehen in einer 1:1 Beziehung. Außerdem sind sowohl die Attribute von Projekt, als auch die von Projektplan funktional abhängig von

Tatsächlich können auch sie zusammengefasst werden. Die neue Entität hat zwei Schlüssel Projektnummer und Projektname:

Projekt:

ProjNr	Projektname	EinsatzPLZ	Einsatzort	DatumVon	DatumBis
P1	BMW-MaterialDB	80851	München	1.2.07	31.7.07
P2	Schiffsverleih	20095	Hamburg	1.7.07	31.8.07
P3	BayernLB-DWH	81377	München	1.6.07	31.10.07

Definition 5.10 (Verbundtreue/Wiederholung)
Eine Zerlegung einer Entität in mehrere andere durch Zerlegung in sich überlappende Attributmengen ist *verbundtreu* (oder *verlustfrei*), wenn durch Kreuzproduktbildung (join) entlang gleicher Attribute mit gleichen Werten keine falschen Objekttupel entstehen. □

In unserem Fall muss man hier das Ergebnis der Synthese betrachten. Nur wenn dieses nicht mehr Tupel enthält als die ursprüngliche Entität, ist die Bedingung erfüllt. Das ist z.B. dann der Fall,

- wenn alle überlappenden Attribute identisch sind und
- in den überlappenden Attributen mindestens ein *Schlüssel* der Entität enthalten ist.

Allerdings kann es auch Entitäten geben, die zusammengefasst weredn können, aber obige Eigenschaft nicht abedecken, z.B. wenn es zwei alternative Schlüssel gibt, von denen aber in jeder Entität nur einer enthalten ist.

Siehe Beispiel 5.9!

Zu beachten ist dabei auch, dass im Schlüssel neben eigenen Attributen auch Fremdschlüssel aus anderen Entitäten enthalten sein können.

5.1.6 Erstentwurf des E/R-Modells

In der Praxis ist eine Kombination von Ansätzen zu Dekomposition und Synthese sinnvoll. Dabei wird man einen ersten Wurf für ein ER-Modell aus den in der Anwendungsdomäne verwendeten Geschäftsobjekten gewinnen. Diesen Entwurf kann man schrittweise komplettieren und transformieren bis zum Erreichen der gewünschten Eigenschaften.

In aller Regel wird man also mit einem ersten Modellentwurf beginnen und diesen unter Anwendung der Dekompositions- und Syntheseverfahren stetig verbessern, bis das Modell bestimmte vorgegebene Kriterien erfüllt (z.B. 1NF). Das bedeutet, dass man zunächst einmal einen brauchbaren ersten Wurf für das Modell benötigt, von dem ausgehend die Verbesserung des Modells verfolgt werden kann. Es ist naheliegend, dass von der Güte dieses ersten Entwurfs abhängt, wie schnell und komplikationslos daraus das Zielmodell abgeleitet werden kann.

Daher ist im Folgenden ein Beispiel für ein simples Vorgehensmodell gegeben, mit dem man einen solchen ersten Wurf erhält.

Schritt 1:

- Bilden von *Entitäten* mit Schlüsselattributen
- Festlegen von *Primärschlüsseln*
- Beschreiben von *Parallelschlüsseln*

Schritt 2:

- *Beziehungen, Kardinalität* der beteiligten Entitäten ergänzen
- Definieren von *Attributen*und deren Datentypen

Schritt 3:

- Integritätsbedingungen an *Attribute*
- Integritätsbedingungen an *Beziehungen*

Schritt 4:

- Beschreibung der *Attribute*
- Beschreibung der zugehörigen *Datentypen*

Ergebnis dieser Schritte ist ein Modell, in dem bereits so viel Information hinterlegt ist, dass daraus die für die Normalisierung notwendigen Erkenntnisse gezogen werden können.

Hilfestellungen:

- Ob eine Information Attribut oder Beziehung ist, ist eine *Designentscheidung*.
- Ob eine Information eine eigene Entität ist, ist ebenfalsl eine *Designentscheidung*, die oftmals erst im Lauf der Modellierung klar wird.
- Ein Attribut, das in mehreren Entitäten mit demselben Typ vorkommt, sollte oft zu einer *Entität* gemacht werden.
- Eine Beziehung zu einer Entität, die nur von einer einzigen anderen verwendet wird, kann auch durch ein *Attribut* ersetzt werden.
- Die Entscheidung über eine *Generalisierung* sollte nicht zu früh getroffen werden.
- Redundanzen zu erkennen, setzt immer zuerst das Verständnis des fachlichen Anwendungsbereichs, der *Domäne* voraus.

5.2 Vorgehensmodelle

Im letzten Abschnitt wurden Verfahren beschrieben, die spezifische Empfehlungen über die einzelnen Schritte beim Modellentwurf geben. Häufig sind diese Teil eines umfassenden Vorgehensmodells, z.B. für den gesamten Zyklus der Softwareentwicklung.

Darüber hinaus wird hier eine Einordnung der ER-Modellierung in den Gesamtkontext des Softwareentwicklungsprozesses gegeben. Das Vorgehen bei der ER-Modellierung ist dabei nur ein Ausschnitt aus dem gesamten Vorgehensmodell, wobei Schnittstellen zu anderen Teilen des Prozesses bestehen.

- Eines der ältesten und vor allem im staatlichen Bereich häufig eingesetztes Modell ist das *V-Modell*, das in den letzten Jahren im *V-Modell XT* gründlich reformiert worden ist.

- Im Kontext der objektorientierten Vorgehensmodelle konnte die Firma Rational ihren Rational Unified Process *RUP* sehr gut platzieren.
- Ebenfalls im Windschatten der objektorientierten Softwareentwicklung wurden leichtgewichtigere Vorgehensmodelle entwickelt, wie z.B. eXtreme Programming *XP*, das auch für seine plakativen Konzepte wie z.B. *Pair Programming* berühmt wurde.
- Mit dem Versuch, sich als pragmatische ("agile") Alternative zu mächtigen Vorgehensmodellen zu positionieren, stellte auch die OOSE ihren oose Engineernig Process *OEP* vor.
- Ebenfalls unter der Flagge der agilen Methoden in der Softwareentwicklung segeln iterative und inkrementelle Basisansätze wie etwa die *Testgetriebene Entwicklung*.
- Ein Ansatz aus der jüngeren Zeit ist die Verwendung von *Pattern* zur Darstellung bestimmter Fragmente im Entwicklungsprozess. Dabei könnte z.B. ein ER-Pattern zum Einsatz kommen, in dem immer dann ein ER-Modell aus vorhandenen Ergebnistypen entwickelt wird, wenn eine relationale Datenbank benötigt wird.
- Mehr oder weniger an einen dieser Ansätze angelehnt sind in der Regel unternehmensspezifische Vorgehensmodelle. Häufig stellen diese ein unternehmensweit festgelegtes *Tayloring* eines Ansatzes dar.

Kapitel 6

UML

6.1 Einführung

6.1.1 Allgemeines

Mitte der 90er Jahre gab es diverse Methoden zur objektorientierten Spezifikation. Prominenteste Vertreter waren die Booch-Methode (von Grady Booch), Object Modelling Technique (von James Rumbaugh) und OOSE (von Ivar Jacobson).

Zur Vereinheitlichung hat die Firma Rational alle diese Methodiker angestellt und 1997 erschien die Modellierungssprache UML (Unified Modelling Language). Sie ist mittlerweile beim Industriekonsortium OMG (Object Management Group) standardisiert, seit 2005 in der Version UML 2.0.

Die UML ist vorrangig gedacht für die Erfassung von *objektorientierten Modellen*, d.h. Modellen in denen Daten und Funktionen auf diesen Daten zu Klassen zusammengefaßt werden. Sie eignet sich aber auch zur Datenmodellierung.

6.1.2 Übersicht über die UML 2.0

Das Metamodell der UML 2.0 enthält sowohl Teile für die Erfassung der Statik eines Modells (Klassen, Komponenten, Pakete und ihre Beziehungen) als auch Teile für die Erfassung der dynamischen Abläufe.

Abbildung 1.1 zeigt die Übersicht über die Bestandteile des UML-Metamodells. Es sind im Einzelnen:

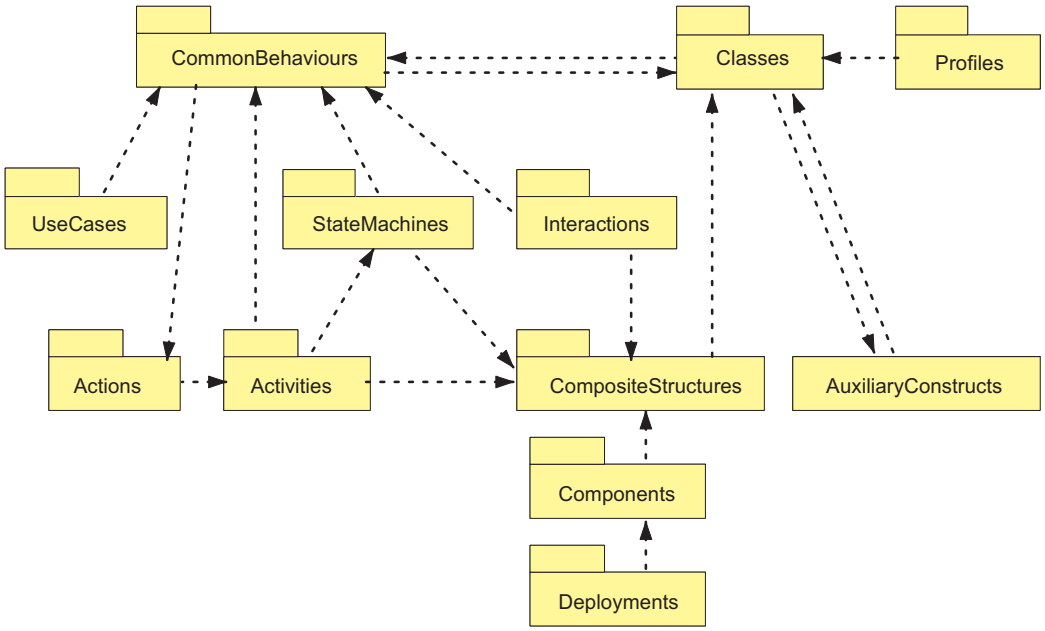


Abbildung 6.1: Übersicht über Bestandteile der UML 2.0 (aus der UML-Norm)

Use Cases: Anwendungsfälle, d.h. die von außen erkennbaren Dienste des Systems für einen externen Nutzer

Classes: Klassen und ihre Beziehungen

Actions, Activities: Konstruktre für Ablaufmodellierung (ähnlich zu Flußdiagrammen)

StateMachines: endliche Automaten (mit Parallelität)

Interactions: Modellierung der Interaktion von Objekten

CommonBehaviours: abstrakte Hilfsklassen für Verhaltensmodellierung

Components: Komponentenmodellierung (zur Modularisierung eines Systems)

Deployments: Konstrukte für die Modellierung der physischen Installation von Software (z.B. auf Rechnerknoten)

CompositeStructures, AuxiliaryConstructs: diverse abstrakte Hilfsklassen

Profiles: Konstrukte zur Anpassung des UML-Metamodells

6.2 Modellierung der statischen Struktur

Definition 6.1 (UML-Notation für eine Klasse)
Eine Klasse wird in der UML über einen Kasten mit mehreren Abteilungen wie folgt dargestellt:

Klassenname
Attribute
Operationen

- Per UML-Konvention ist der Klassenname ein großgeschriebenes Substantiv (wie alle Typnamen).
- Attributdefinition haben folgende Syntax:
  `name : Typ = Anfangswert { Randbedingung }`

  Hierbei ist `name` der Name des Attributs, `Typ` ein Elementartyp oder eine Klasse, `Anfangswert` der anfängliche Wert für dieses Attribut und `Randbedingung` eine Anforderung an den Attributwert in Abhängigkeit von anderen Eigenschaften des entsprechenden Objekts.

  Optional sind alle Teile außer dem Attributnamen.
- Operationen haben folgende Syntax:
  `name ( parameterliste ) : Typ { Randbedingung }`

  Hierbei ist `name` der Name der Operation, `Typ` ein Elementartyp oder eine Klasse und `Randbedingung` eine Anforderung an die Parameter oder das Ergebnis der Operation. Die Parameterliste ist eine Folge von Namen und Typen in Ada-Syntax (siehe Beispiel).

  Optional sind alle Teile außer dem Namen und den Parameterklammern.
- Es gibt (neben anderen) weitere Beifügungen für ein Attribut oder eine Operation: dessen Sichtbarkeit (`+` = "öffentlich", `#` = "in abgeleiteten Klassen sichtbar" und `-` = "privat") und die Tatsache, dass es abgeleitet ist (= `/`). Dies steht jeweils vor dem Namen der Eigenschaft. Die Namen von *Klasseneigenschaften* sind unterstrichen.
- Per UML-Konvention werden Attribut- und Operationsnamen kleingeschrieben. Attributnamen sind typischerweise Substantive, Operationsnamen Verben.

□

Beispiel 6.2 (Notation einer UML-Klasse)
Unser Buchautor könnte wie folgt modelliert werden:

Buchautor
+name : Zeichenkette { name > " }
+vorname : Zeichenkette
-geburtsdatum : Datum
-adresse : Adresse
+alter(**in** datum : Datum) : Ganzzahl {datum ≥ geburtsdatum}
+anlegen(**in** name : Zeichenkette)

Definition 6.3 (UML-Notation für eine Assoziation)
Eine *Assoziation* wird in der UML durch Liniensegmente zu in Beziehung stehenden Klassen von einer Raute aus dargestellt (bei der Raute steht auch der Beziehungsname). Bei binären Assoziationen entfällt die Raute normalerweise.

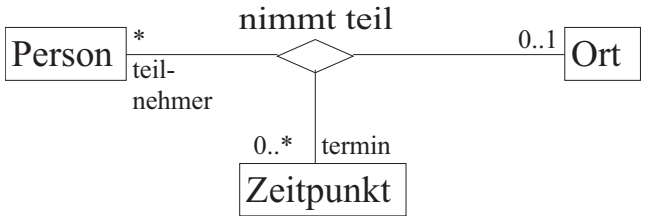
An den Enden wird je beteiligter Klasse folgendes notiert:

- ein Rollenname (ähnlich einem Attribut),
- eine Kardinalität (beliebige Intervalle von Ganzzahlen durch Komma getrennt, die Intervallgrenzen sind durch zwei Punkte verbunden),
- zusätzliche Beifügungen (siehe unten)

Eine Leserichtung kann durch ein Dreieck beim Assoziationsnamen angegeben werden.

Ein Assoziationsname ist oft eine Satzphrase (ein Satz ohne Subjekt und Objekt), Rollennamen sind kleingeschriebene Substantive. Wenn der Rollenname fehlt, ist der kleingeschriebene Klassenname der Rollenname. □

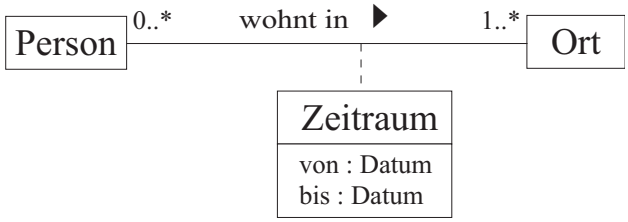
Beispiel 6.4 (Notation einer UML-Assoziation)
Wir greifen nochmals die Beziehung "nimmt teil" auf. Die UML-Notation ist sehr ähnlich zu der E/R-Notation.



Bei der binären Beziehung entfällt die Raute:

Definition 6.5 (Notationsvarianten bei Assoziationen)

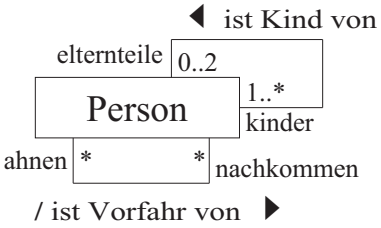
Assoziationstyp: Assoziationen können Informationen zu einer Verbindung zwischen Objekten tragen; dies wird als Klasse modelliert und mit der Assoziation über eine Strichlinie verbunden.



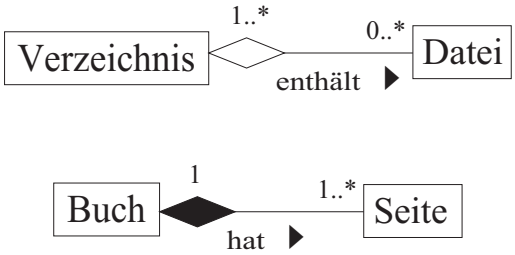
qualifizierte Assoziation: Wenn in einer Assoziation der Zugriff auf bezogenes Objekt nur über ein Schlüssel möglich ist, kann man das am Assoziationsende vermerken. Man schreibt das Schlüsselattribut in ein Kästchen.

Auffällig ist, dass sich die Kardinalität (im Vergleich zu Beispiel 1.4) ändert: Eine Person ist Autor von höchstens einem Buch mit einer bestimmten ISBN.

abgeleitete Assoziation: Wenn sich eine Assoziation aus anderen Beziehungen ergibt, dann wird vor den Assoziationsnamen ein '/' geschrieben.



Aggregation, Komposition: Wenn in einer binären Assoziation eine Klasse Teil der anderen ist und auch von seiner Lebensdauer davon abhängt, aber zu mehreren gehören darf, dann wird dafür die *Aggregation* verwendet (eine nichtausgefüllte Seite am Assoziationsende der umfassenden Klasse). Wenn eine Klasse exakt zu einer umfassenden gehört, nimmt man eine *Komposition* (mit einer gefüllten Raute).



□

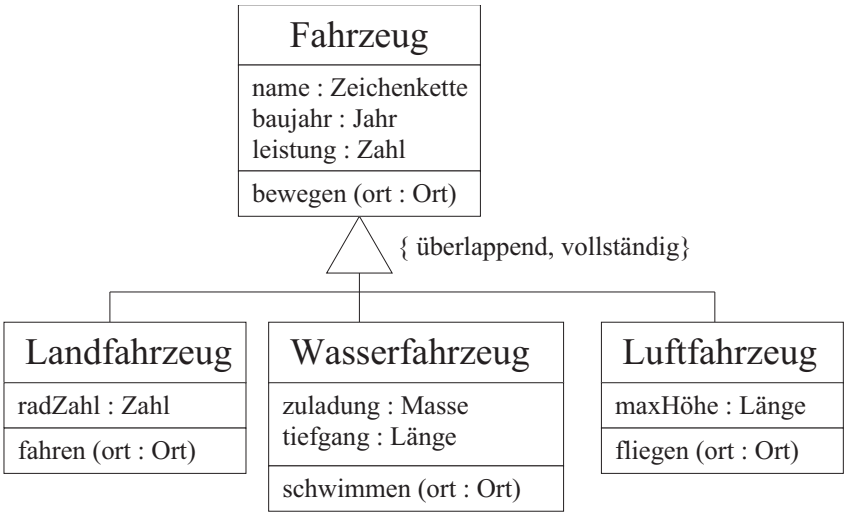
Definition 6.6 (Generalisierung)
Wenn eine Klasse ein Spezialfall einer anderen Klasse ist, kann das durch eine *Generalisierungsbeziehung* ausgedrückt werden (eine Beziehung mit Dreieckspfeil am Ende der allgemeinen Klasse).

Sehr vereinfacht (!!) muss für die Klassen S (Spezialfall) und A (allgemeine Klasse) gelten, dass

- die Attribute von S Obermenge von A sind und die entsprechenden Attributtypen identisch sind,
- die Operationen in S eine Obermenge von A sind und die entsprechenden Signaturen identisch sind und
- sämtliche Randbedingungen aus A auch in S gelten.

Für Spezialisierungen in mehrere Unterklassen kann man angeben, ob sie überlappend oder disjunkt bzw. vollständig oder unvollständig sind (durch eine Randbedingung in geschweiften Klammern beim Generalisierungsdreieck). □

Beispiel 6.7 (Generalisierung)
Ein Fahrzeug ist entweder ein Wasserfahrzeug, Landfahrzeug oder Luftfahrzeug. Die Aufteilung ist also vollständig, aber sicherlich überlappend, da es auch Kombinationsfahrzeuge gibt.


Fahrzeug
name : Zeichenkette
baujahr : Jahr
leistung : Zahl
bewegen (ort : Ort)
{ überlappend, vollständig}
Landfahrzeug
radZahl : Zahl
fahren (ort : Ort)
Wasserfahrzeug
zuladung : Masse
tiefgang : Länge
schwimmen (ort : Ort)
Luftfahrzeug
maxHöhe : Länge
fliegen (ort : Ort)

Kapitel 7

ER-Modellierung mit der UML

7.1 Einführung

7.1.1 Allgemeines

Um mit den Mitteln der UML sinnvoll ein ER-Modell darzustellen, muss die UML

- einerseits eingeschränkt werden auf die in einem ER-Modell verwertbaren Konstrukte
- und andererseits erweitert werden um die für ein ER-Modell unverzichtbaren Elemente, die in der UML zunächst nicht vorgesehen sind.

Im Folgenden wird die Verwendung der UML für die ER-Modellierung schrittweise aufgebaut. Dabei muss unterschieden werden zwischen der Darstellung eines logischen Modells und der Darstellung eines physischen Modells. Die Darstellung eines logischen Modells in der UML liegt näher an der allgemeinen Verwendung der UML-Klassenmodellierung und erfordert daher weniger Spezialkonstrukte als die Darstellung eines technischen Modells. Es gibt in der UML aber sowohl Nutzungskonzepte für die logische Modellierung als auch für die physische Modellierung.

Um ein solches Nutzungskonzept in der UML umsetzen zu können, bietet die UML selbst einen Mechanismus, der das verwendete Metamodell speziell dem jeweiligen Nutzungskonzept anpasst, das *UML-Profil*.

7.1.2 UML-Profile

UML-Profile sind selbst auch Elemente der UML. Ein konkretes UML-Profil stellt eine Erweiterung des UML-Metamodells dar, die folgende Anteile enthalten kann:

- Einschränkung des Metamodells auf eine Teilmenge der möglichen Elemente,
- Erweiterung des Metamodells um zusätzliche Konstrukte, die in Bezug zu vorhandenen Elmenten gesetzt werden,
- Umdefinition bestimmter Elemente des Metamodells, wie z.B. um die Darstellung zu verändern
- Belegung abstrakter Elemente des Metamodells mit spezifischen Werteausprägungen.

In der Denkweise der objektorientierten Modellierung könnte man das UML-Profil als eine "Spezialisierung" des UML-Metamodells bezeichnen, die sowohl Erweiterung, Einschränkung als auch Umdefinition von Metamodellelementen zulässt.

Definition 7.1 (UML-Notation für ein Profil)
Ein UML-Profil wird als Paket mit dem Schlüsselwort ≪profile≫ dargestellt. Ein Profil ist eine Spezialisierung von Paket und umfasst zwei Arten von Elementen:

- eine Menge von Stereotypen, das heißt elementaren Erweiterungen des Metamodells der UML2, und
- eine Menge von Paketimporten für Pakete aus dem Metamodell der UML2

□

Die entscheidenden Elemente eines Profils sind die elementaren Erweiterungen des Metamodells der UML2 repräsentiert als Stereotype. Wir werden später in Bild 7.3 sehen, wie ein Profil als UML-Modell dargestellt wird.

Profile werden im allgemeinen auf Pakete, meistens aber auf Modelle (als Spezialisierung von "Paket"), angewendet. Es ist also sowohl möglich, innerhalb eines allgemeinesn UML-Modells ein Paket für die Datenmodellierung

vorzusehen und darauf das ERM-Profil anzuwenden, als auch ein gesamtes Modell für die ER-Modellierung vorzusehen.

Die Anwendung eines Profils auf ein Paket ist eine Spezialisierung des Paketimports (gekennzeichnet durch das Stereotyp "apply"). Sie führt zu den beiden erwünschten Effekten:

- Erstens importiert das Paket alle Elemente aus dem Profil. Falls das Profil zum Beispiel einen Aufzählungstyp "Farbe" definiert, kann das Paket "Farbe" als Attribut-Typ verwenden.
- Zweitens kann das Paket alle Stereotype einsetzen, die im Profil definiert sind.

Eine Profilanwendung wird als gestrichelte Linie mit offener Pfeilspitze auf der Seite des Profil gezeichnet. Sie unterscheidet sich von anderen Abhängigkeitsbeziehungen durch das Schlüsselwort ≪apply≫.

7.2 Anwendungsfälle für den Einsatz der UML zur ER-Modellierung

Im folgenden sind die verschiedenen Kategorien zum Einsatz der UML in der Datenmodellierung dargestellt. Dabei ist die Komplexität der Anforderungen aufsteigend.

- Anwendungsfall 1: Verwendung der UML-Notation für ein reines ER-Modell. Hierzu benötigen wir für jedes Konstrukt des ER-Modells ein Äquivalent in UML. Da die Entitäten letztlich in eine relationale Datenbank überführt werden, wird damit eine Transformation in ein ER-Modell vorbereitet.

  Denkbar ist dieser Anwendungsfall z.B. wenn die Methode ERM eingesetzt werden soll, dazu aber ein bereits genutztes UML-Werkzeug verwendet werden soll.

- Anwendungsfall 2: Verwendung der UML-Notation für ein Klassenmodell, in das die zu persistierenden Klassen eingebettet sind.

  Denkbar ist dieser Anwendungsfall z.B. bei objektorientierter Analyse in einem Projekt, wo die Datenmodellierung einerseits integraler Bestandteil der Klassenmodellierung sein soll, aber andererseits die Daten relational gespeichert werden sollen.

- Anwendungsfall 3: Verwendung der UML-Notation für ein technisches Datenbankmodell. Diese Anforderung geht über den bisher gezeigten Umfang hinaus, da ein technisches Datenbankmodell zusätzliche Informationen zur physischen Umsetzung beinhalten muss. Ein UML-Profil für ein technisches Datenbankmodell wäre ein typisches Beispiel für den Einsatz der Profile zur MDA (Model driven Architecture). Dabei würde aus einem logischen Modell über Anreicherungen und Transformationen eine technische Repräsentation abgeleitet werden. $\rightsquigarrow$ Nächste Vorlesung!

- Anwendungsfall 4: Verwendung der UML-Notation für ein logisches ER-Modell, in dem aber bereits alle fachlichen Informationen enthalten sind, die für die Ableitung eines technischen Datenbankmodells gebraucht werden. Auch diese Anforderung ist typisch für einen generativen Ansatz in der Softwareentwicklung und zielt darauf ab, eine Trennung der fachlichen von den technischen Modellen sicherzustellen und trotzdem die Umsetzung in die Technik vorzubereiten.

7.2.1 Darstellung der ER-Konstrukte in der UML

Ziel des Folgenden ist es, den ersten Anwendungsfall abzudecken, d.h. ein ER-Modell mit den Ausdrucksmitteln der UML darzustellen. Dazu wird ein Profil vorgeschlagen, das dies leistet. Es gibt aber wie generell in der Modellierung auch hier beliebig viele andere Profile, die ebenso richtig sind.

Grundkonzepte

Die drei grundlegenden Konzepte sind folgende:

entity (Entität): Typ von Objekten der realen oder Vorstellungswelt, die für das Informationssystem bedeutsam sind (z.B. ein Buch, eine Person...);

Die Entität wird in der UML durch eine Klasse dargestellt.

relationship (Beziehung): Beziehung zwischen Entitäten (z.B. eine Person ist Autor eines Buchs);

Die Relationship ist durch die Assoziation korrekt wiedergegeben.

Attribut: datenhaltige Eigenschaft von Entitäten oder Beziehungen (z.B. das Erscheinungsdatum eines Buchs);

Das Attribut findet sich in der UML direkt wieder.

Bemerkung: Diese Festlegung bezieht sich auf die in der gesamten Vorlesung verwendete Interpretation der "Entität" als Klasse, nicht als Objekt. In der Terminologie, die dem Begriff Entität für die Instamz verwendet, wäre also hier der "Entitätstyp" als Klasse repräsentiert.

Die obigen Konzepte werden im UML-Modell wie folgt graphisch dargestellt:

- Entitäten durch ein Rechteck mit bis zu drei Sektionen, in dessen oberem Segment der Namen steht (siehe beispielsweise "Professor" in Bild 7.1);
- Relationen durch eine Verbindungslinie zu sämtlichen beteiligten Klassen(siehe beispielsweise "liest" in Bild 7.1).
- Attribute stehen in der zweiten Sektion der Klasse; wenn der Datentyp des Attributs bekannt ist, folgt er dem Attributnamen und einem Doppelpunkt (siehe beispielsweise "Titel" von "Vorlesung" in Bild 7.1).

So weit konnten die Elemte der ER-Modellierung einfach auf vorhandene Elemente der Klassenmodellierung überführt werden. Das funktioniert nicht bei den spezifischen ER-Konstrukten, so dass hierfür spezielle Konstrukte eingeführt werden:

Stereotype

Schlüsselattribute sind in der UML unbekannt, da jedes Objekt per se (durch seine Object-ID) identifizierbar ist. Daher muss für die Kennzeichnung von Schlüsselbestandteilen eine Sonderkonstruktion geschaffen werden.

- Im E/R-Diagramm werden Primärschlüsselattribute unterstrichen. Im UML-Diagramm werden Fachliche Primärschlüsselattribute mit Hilfe des Stereotyps ≪PK≫ (Primary Key) gekennzeichnet (siehe beispielsweise "ISBN-Nr." von "Buch"in Bild 7.1).
- Analog werden Relationen, die zur Identifikation beitragen mit demselben Stereotyp ≪PK≫ gekennzeichnet (siehe beispielsweise "hat Bestand" in Bild 7.1);

- Wenn es in einer Klasse neben dem Primärschlüssel noch weitere n Parallelschlüssel gibt, so werden die Attribute, die zu einem der alternativen Schlüssel beitragen, mit Hilfe des Stereotyps ≪PAR1≫ bis ≪PARn≫ gekennzeichnet. Im E/R-Diagramm gibt es für Sekundärschlüsselattribute keine Notation. (siehe beispielsweise "InventarNr" in Bild 7.1);

- Analog werden Relationen, die zur Identifikation in einem Parallelschlüssel beitragen, mit demselben Stereotyp ≪PAR1≫ bis ≪PARn≫ gekennzeichnet;

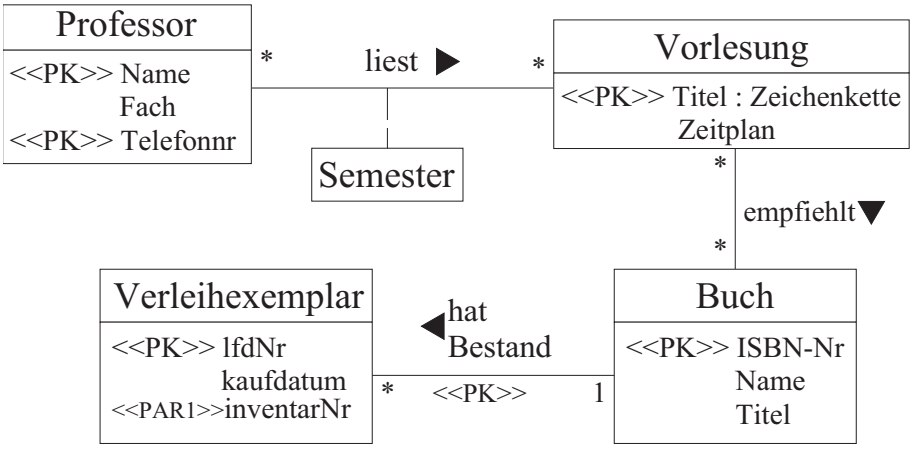


Abbildung 7.1: Beispiel eines UML-Datenmodells

Weitere Elemente der E/R-Modellierung

Eigenschaften von Attributen Im Folgenden sind die in der ER-Modellierung eingeschlossenen Konstrukte jeweils einem passenden UML - Konstrukt gegenübergesetellt.

Datentypen: Im E/R-Diagramm werden die atomaren Datentypen hinter dem Namen des Attributes (nach einem Doppelpunkt) notiert. Die Notation bei der UML-Modellierung ist die selbe.

zusammengesetzte Attribute: Attribute können anstatt nur aus einem atomaren Typ auch zusammengesetzt sein aus Bestandteilen mit beliebigem Typ.

Im E/R-Diagramm werden die Bestandteile eines zusammengesetzten Attributs mit diesem verbunden wie mit einer Entität. In der UML-Modellierung werden zusammengesetzte Attribute als Instanz einer anderen Klasse dargestellt.

abgeleitet: Wenn ein Attribut sich durch Berechnung aus anderen ergibt, dann bezeichnet man es als *abgeleitet* (typisches Beispiel: Alter einer Person).

Im E/R-Diagramm werden abgeleitete Attribute durch gepunktete Ovale notiert. Im UML-Diagramm wird das abgeleitete Attribut durch einen vorangestellten Schrägstrich gekennzeichnet.

einwertig/mehrwertig/optional: Da der Regelfall der ist, dass ein Attribut genau einen Wert seines Attributtyps annimmt, es also *einwertig* ist, wird dies für ein Attribut unterstellt, solange nichts anderes explizit ausgesagt wird. Kann das Attribut eine Menge von Werten seines Typs haben, ist es also *mehrwertig*, so wird dies durch Angabe der Kardinalität hinter dem Namen des Attributes ausgewiesen. Wenn es keinen Wert haben darf, dann ist es *optional*, was in der UML durch das Stereotyp "nullable" bzw. durch die Kardinalität 0..1 dargestellt wird.

Eigenschaften von Beziehungen

Kardinalität/Optionalität: Die *Kardinalität* einer Beziehung wird an jedem Ende der Beziehung durch Angabe eines Intervalls dargestellt.

In der UML wird dieses Intervall in der Form "Untergrenze..Obergrenze" dargestellt, wobei * für eine beliebige Zahl (n) steht.

Generalisierung/Spezialisierung (IST-Beziehung): bzw. Vererbung. Diese wird im UML-Modell als ein Pfeil dargestellt, an dessen Ende eine Pfeilspitze in Form eines nicht ausgefüllten Dreiecks auf die Generalisierung deutet.

binäre vs. n-äre Beziehungen: Auch Assoziationen im UML-Modell können statt zwischen zwei auch zwischen n Klassen verlaufen. (Beispiel: "Person" hat "Termin" in "Raum").

m:n-Beziehungen/Auflösung in eigener Entität: Auch im UML-Modell kann die Auflösung einer m:n-Beziehung in einer eigenständigen Auflöserklasse vollzogen werden. Es ergeben sich keine Besonderheiten in der Notation.

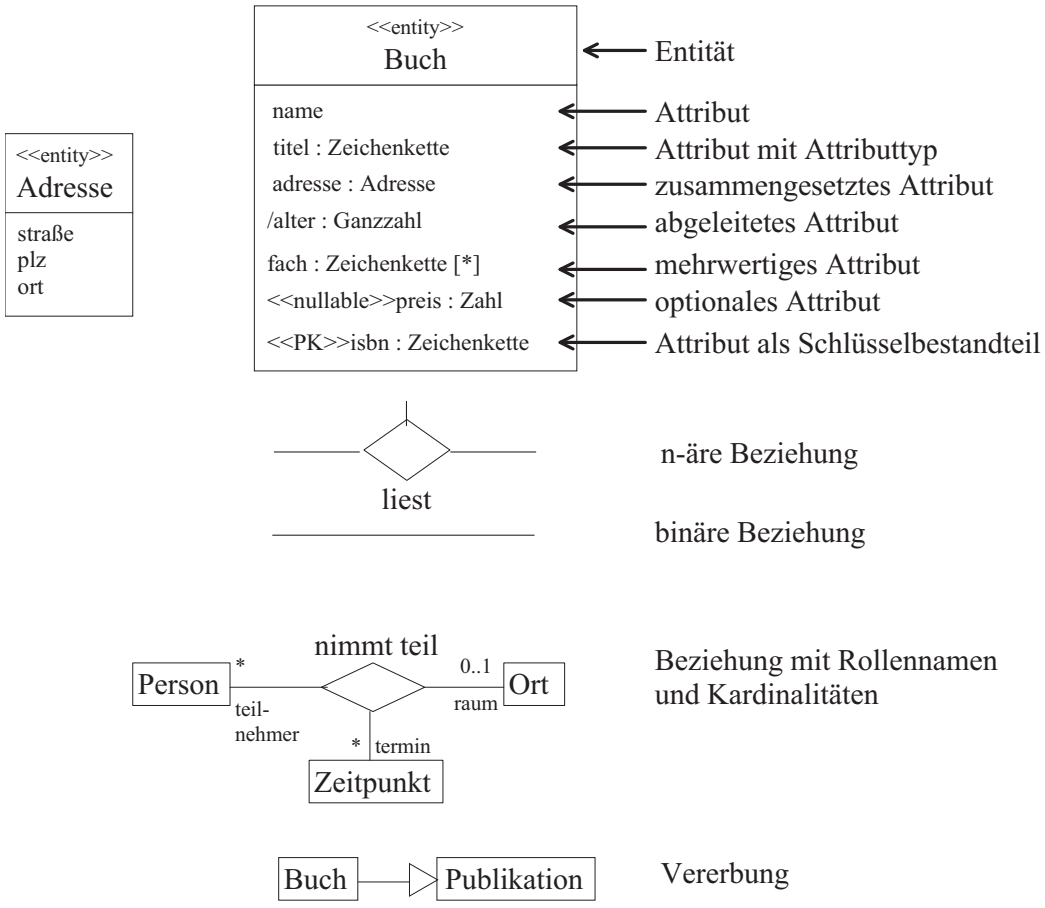


Abbildung 7.2: graphische Notationen für die Datenmodellierung in UML

7.2.2 Einbettung der ER-Modellierung in ein OO-Modell

In diesem Kapitel wird der zweite Anwendungsfall bearbeitet, d.h. die Verwendung der UML-Notation für ein Klassenmodell, in das die zu persistierenden Klassen eingebettet sind.

Grundkonzepte

Im nächsten Schritt wird nun das im letzten Kapitel gewonnene Profil weiter ergänzt, um den zweiten Anwendungsfall abzudecken, d.h. diese Konstrukte der ER-Modellierung sollen nun in einem allgemeinen OO-Modell verwendet werden. Dazu ist es nötig, die speziellen ER-Teile innerhalb des Gesamtmodells kenntlich zu machen.

Auch weiterhin werden keine zusätzlichen UML-Konzepte für die ER-Teile des Modells genutzt. Die Aufgabe beschränkt sich also darauf, in einem Klassenmodell die relevanten Elemente auszuzeichenen. Dies erreicht man durch Hinzunahme weniger weiterer Stereotype.

Zusätzlich benötigte Stereotype

Entität: Um unter allen Klassen des UML-Modelles diejenigen auszuzeichnen, die Teil des darin eingebetteten UML-Modelles sind, erhalten alle diese Klassen das Stereotyp ≪Entity≫.

Attribute: Die Attribute einer Klasse, die als Entität gekennzeichnet ist, werden mit dem Stereotyp ≪persistent≫ als relevant für die Datenmodelierung markiert. Attribute von nicht als Entität gekennzeichneten Klassen können nicht als persistent gekennzeichnet werden.

Beziehungen: Die Beziehungen, die Teil der Datenmodellierung sind, werden ebenfalls mit dem Stereotyp ≪persistent≫ gekennzeichnet. Auch hier können persistente Beziehungen nur zwischen Entitäten auftreten, nicht zwischen anderen Klassen.

7.2.3 Verwendung der UML-Modellierung für ein technisches Datenmodell

In diesem Kapitel wird der dritte Anwendungsfall bearbeitet, d.h. die Verwendung der UML-Notation für ein technisches Datenbankmodell. Diese An-

forderung geht über den bisher behandelten Umfang der ER-Modellierung hinaus, da ein technisches Datenbankmodell zusätzliche Informationen zur physischen Umsetzung beinhalten muss.

Ein UML-Profil für ein technisches Datenbankmodell wäre ein typisches Beispiel für den Einsatz der Profile zur MDA (Model driven Architecture). Dabei würde aus einem logischen Modell über Anreicherungen und Transformationen eine technische Repräsentation abgeleitet werden.

Die Einführung einer Transformation vom logischen ins technische Datenmodell ist Gegenstand der nächsten Vorlesung - Dieses Kapitel beinhaltet hier nur die Vorstellung eines UML-Profiles für ein technisches Datenbankschema.

Beispiele für weitere Konstrukte:

Datenbank: Im technischen Modell kommen erstmals neben den logischen Bestandteilen des UML-Klassendiagrammes technische Repräsentanten der UML ins Spiel. Datenbanken werden als UML-Komponenten mit dem Stereotyp ≪Database≫ gekennzeichnet. Oft wird an das Stereotyp eine besondere Darstellung in den Diagrammen gekoppelt.

Schema: Die logische Einheit Schema wird sinnvoll in einem UML-Paket dargestellt. Diese besonderen Pakete werden durch das Stereotyp ≪Schema≫ kenntlich gemacht.

Tabelle: Im technischen Modell sollen nicht mehr Entitäten sondern Tabellen dargestellt werden. Diese werden als Klassen mit Hilfe des Stereotyps ≪Table≫ gekennzeichnet.

Spalten: Die Spalten der Tabelle sind Attribute der als ≪Tabellen≫ markierten Klassen und werden mit dem Stereotyp ≪Column≫ gekennzeichnet.

Schlüssel: Im technischen Modell werden die Schlüssel zur Anlage von Primär- und Sekundärindizes verwendet. Daher benötigen wir auch hier das Stereotyp ≪PK≫ bzw. ≪PAR≫. Gleichzeitig kann dieser Stereotyp auch zur Kennzeichnung von Constraints zu den Schlüsseln verwendet werden.

Foreign Key: Im technischen Modell müssen aus den Beziehungen des logischen Modelles Fremdschlüsselattribute werden. Diese besonderen Attribute werden durch das Stereotyp ≪FK≫ gekennzeichnet. („Migrated Attributes“)

Mengengerüste: Mengengerüste können z.B. im technischen Modell zur Kalkulation benötigten Speicherplatzes verwendet werden. Darauf aufbauend können Festlegungen für benötigte Speicherbereiche getroffen werden.

Verantwortlichkeiten/Kostenstellen: Unter Umständen werden für die Einrichtung der Datenbank eine Kostenstelle und Verantwortliche benötigt. Auch diese Informationen werden oft im Rahmen der fachlichen Analyse gewonnen.

Das Profil für Datenbankmodellierung sieht als UML-Modell aus wie in Bild 7.3 dargestellt.

- Die Metamodellklassen "Package", "Component", "Class", "Property" und "Association" werden aus der UML aus deren Paketen "Classes" und "Components" importiert, da die Stereotype sich darauf beziehen.
- Die Beziehungen mit ausgefüllten Spitzen sind UML-Erweiterungen ("Extensions"). Sie verlaufen von Stereotyp zur Metaklasse aus dem zu ergänzenden Metamodell.
- Stereotype können für eine Metaklasse optional oder zwingend sein.
- Wenn ein Stereotyp ein Attribut hat, muss im Modell beim entsprechenden Modellelement ein Wert für dieses Attribut hinterlegt werden (also z.B. bei eine Klasse mit Stereotyp "Tabelle" ein konkrete Zahl der Sätze für "recordCount").

7.2.4 Erweiterung des logischen Datenmodells

Hier geht es um die Umsetzung des vierten Anwendungfalles, also die Verwendung der UML-Notation für ein logisches ER-Modell, in dem bereits alle fachlichen Informationen enthalten sind, die für die Ableitung eines technischen Datenbankmodells gebraucht werden. Auch diese Anforderung ist typisch für einen generativen Ansatz in der Softwareentwicklung und zielt darauf ab, eine Trennung der fachlichen von den technischen Modellen sicherzustellen und trotzdem die Umsetzung in die Technik vorzubereiten.

Zusätzliche Eigenschaften der Klasse, Beziehung oder des Attributs können in zusätzlichen Eigenschaften untergebracht werden. Dabei bewegen wir uns aber weiterhin im Rahmen der logischen Modellierung!

Beispiele für Zusatzeigenschaften:

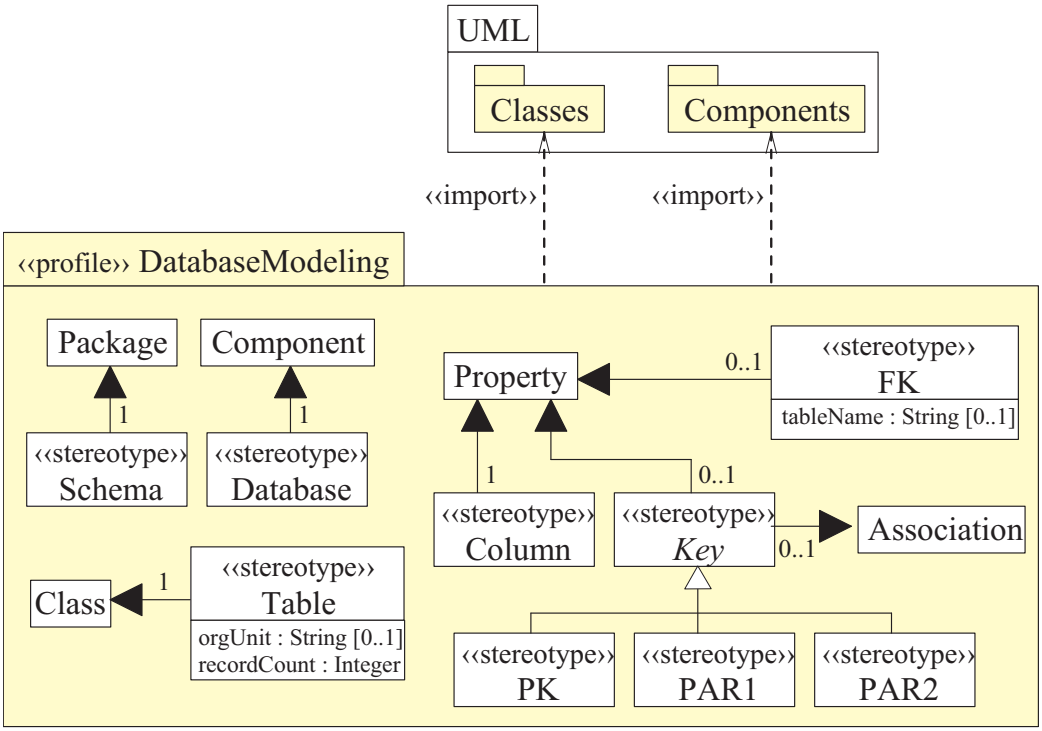


Abbildung 7.3: Erweiterung der UML um ein Datenbankmodellierungsprofil

Löschregeln: Um im technischen Modell und später in der Datenbank eine Regelung für das Löschen abhängiger Objekte zu implementieren ist es sinnvoll, bereits zuvor in der fachlichen Modellierung diesbezüglich eine Entscheidung zu treffen. Diese Festlegung ist z.B. sinnvoll in einem tagged value untergebracht.

Mengengerüste: Mengengerüste können z.B. im technischen Modell zur Kalkulation benötigten Speicherplatzes verwendet werden. Darauf aufbauend können Festlegungen für benötigte Speicherbereiche getroffen werden.

Namensbestandteile: Namen in technischem Modell und Datenbank unterscheiden sich häufig von denen des fachlichen Modells, z.B. auf Grund von Längenrestriktionen. Dafür können bereits im logischen Modell Kurznamen in tagged values festgehalten werden.

Verantwortlichkeiten/Kostenstellen: Unter Umständen werden für die Einrichtung der Datenbank eine Kostenstelle und Verantwortliche

benötigt. Auch diese Informationen werden ft im Rahmen der fachlichen Analyse gewonnen.

Die Sinnhaftigkeit dieser Informationsgewinnung erschließt sich bei Kenntnis der Transformationen vom logischen in ein technisches Datenmodell, wie im folgenden Kapitel dargestellt. Weitere Beispiele können ebenfalls daraus abgeleitet werden.

7.3 Abbildung von UML-Modellen in ein technisches Datenbankmodell

Die Abbildung von statischen UML-Klassenmodellen in ein relationales Modell ist verhältnismäßig geradlinig. Einzige Komplikation bietet die Generalisierung, da für sie mehrere Abbildungsvarianten existieren.

Zusammengefaßt gilt folgendes:

- Klassen und Assoziationen werden in Tabellen abgebildet.
- Die einzelnen Attribute bilden Spalten in den Tabellen einer Klasse oder eines Assoziationstyps.
- Methoden werden nicht abgebildet. Die Bindung an Objekte muß über Mechanismen der Programmiersprache beziehungsweise des dazugehörenden Laufzeitsystems abgedeckt werden.
- Die Benennung von Tabellen und von Spalten kann sich prinzipiell orientieren an den Namen der Klassen und Attribute, sofern nicht technische Randbedingungen dagegen sprechen. Rollennamen können analog als Spaltennamen von Assoziationstabellen verwendet werden.
- Randbedingungen (wie beispielsweise Kardinalitäten) sollten in Integritätsbedingungen der Datenbank übersetzt werden. Meistens muss aber die Zugriffschicht selbst prüfen, ob durch eine Operation die Konsistenz des Objektnetzes verletzt wird oder nicht.

Regel 7.2 (Abbildung von UML-Klassen)
Klassen werden auf eine oder mehrere Tabellen abgebildet und die Attribute einer Klasse auf Spalten in diesen Tabellen (siehe Attributabbildung 7.3). Sofern eine Klasse keine Ober- und Unterklassen hat, wird sie auf genau eine Tabelle im relationalen Modell abgebildet. Ist die Klasse in eine Hierarchie eingebettet, gibt es mehrere Optionen (siehe 7.5) □

Regel 7.3 (Abbildung von UML-Attributen)
Die Attribute einer Klasse werden wie folgt in die Spalten der Klassentabelle abgebildet:

Attributtyp ist skalar: Das Attribut wird (bis auf Spezialfälle wie beispielsweise eine Wertebereichsüberschreitung) auf eine Spalte der Typtabelle abgebildet.

Attributtyp ist zusammengesetzter Typ: Dann werden (bis auf Spezialfälle wie beispielsweise "Datum") die Komponenten des zusammengesetzten Typs rekursiv auf Spalten der Tabelle abgebildet, bis man auf skalare Typen trifft.

Attributtyp ist Liste/Menge: Für das Attribut wird in der Regel eine weitere Tabelle angelegt, die eine 1:n Abbildung zwischen einem Objekt A mit diesem Attribut und den gesammelten Objekten G realisiert. Eine Zeile dieser Sammlungstabelle enthält einen Fremdschlüssel auf das Objekt A und einen Fremdschlüssel für ein Objekt von G. Ist die Anzahl der Partnerobjekte beschränkt, kann auch denormalisiert werden.

abgeleitetes Attribut: Abgeleitete Attribute werden normalerweise über die Zugriffsschicht beim Zugriff auf das Objekt berechnet und nicht gespeichert.

□

Regel 7.4 (Abbildung von UML-Assoziationen)
Assozationen werden normalerweise in eine Tabelle abgebildet, die Tupel aus Fremdschlüsseln auf die beteiligten Klassentabellen enthält.

Bei einer zweistelligen Assozation prüft man, ob zwischen Klasse A und B eine 1:1-Beziehung vorliegt oder eine 1:n-Beziehung, bei der das eindeutige Objekt nicht effizient zu den Partnerobjekten navigieren muss. In diesem Fall kann die Klassentabelle der eindeutigen Klasse auch einen Fremdschlüssel auf die andere enthalten und die Assoziationstabelle kann entfallen. Für eine optionale Beziehung kann das auch so passieren; dann sind die Fremdschlüsselattribute "nullable".

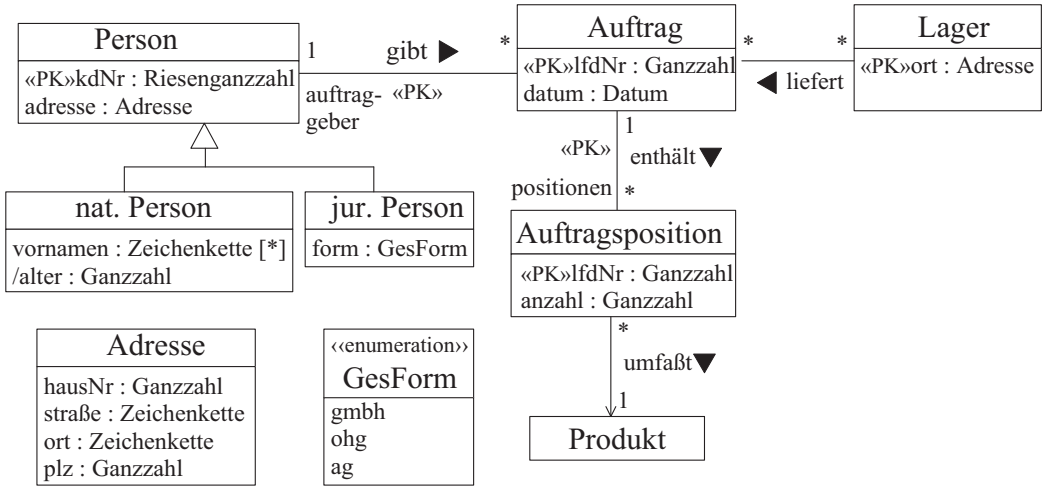
Attribute des Assozationstyps werden bei einer 1:1-Assozation bei einem Partnerobjekt abgelegt, bei einer asymmetrischen 1:n-Assozation beim nichteindeutigen Objekt. Wird eine Assozationstabelle verwendet, sind die Assozationsattribute Spalten der Assozationstabelle. □

Regel 7.5 (Abbildung von UML-Generalisierungen)
Für die Auflösung von Generalisierungsbeziehungen gibt es mehrere Abbildungsmöglichkeiten:

1. Für alle Klassen in der Hierarchie wird insgesamt eine Tabelle benutzt: Die Einträge sind mit einem Typkennzeichen versehen, aber alle Exemplare der Wurzelklasse und sämtlicher Unterklassen stehen in der Tabelle. Für die neuen Attribute jeder Klasse werden entweder
   (a) eigene Spalten in der Tabelle vorgesehen oder
   (b) bei unabhängigen Klassen Spalten überlagert.
2. Jede Klasse in der Hierarchie erhält eine eigene Tabelle. Die Aufteilung der Attribute und Objekte kann wie folgt geschehen:
   (a) Ein Objekt wird nur in einer Tabelle gespeichert, die zu seiner Klasse gehört.
   (b) Ein Objekt wird für jeden Vorfahren in dessen Tabelle gespeichert; die Tabellen erhalten jeweils *sämtliche* Attribute der entsprechenden Klasse.
   (c) Ein Objekt wird für jeden Vorfahrtyp in dessen Tabelle gespeichert; die Tabellen erhalten nur die in diesem Typ neu definierten Attribute.

□

Beispiel 7.6 (Abbildung eines UML-Modells in ein Datenbankmodell)
Mit den obigen Techniken möchten wir folgendes UML-Modell mit fachlicher Kennzeichnung der Primärschlüssel in ein relationales DB-Modell abbilden:



Es werden zwei Entwurfsentscheidungen gefällt:

- Alle Klassen außer “Adresse”, “GesForm” und “Produkt” werden als Tabellen repräsentiert; die ausgenommenen werden expandiert.
- Für die Generalisierung bei “Person” wird als Entwurfsentscheidung angenommen, dass Lesen nach Typ häufig passiert, aber Einfügen relativ selten. Daher wird die Variante gewählt, bei der alle Attribute in den Tabellen der beteiligten Typen abgelegt sind (Variante 2b).

Dann werden die einzelnen Schritte des Verfahrens wie folgt durchgeführt:

- Die Attribute werden — gemäß Regel 7.3 — wie folgt abgebildet:

Attribut	**Art**	**Vorgehen**
Person::kdNr	skalar, groß	aufteilen
Person::adresse	zusammengesetzt	“Adresse” expandieren
nat. Person::vornamen	mehrwertig	in eigene Beziehungstabelle
nat. Person::alter	abgeleitet	weglassen & in Zugriffsschicht
jur. Person::form	Aufzählung	Zeichenkette
Auftrag::lfdNr	skalar	skalar in Datenbank
Auftrag::datum	zusammengesetzt	skalar in Datenbank
Auftragsposition::lfdNr	skalar	skalar in Datenbank
Auftragsposition::anzahl	skalar	skalar in Datenbank
Lager::ort	zusammengesetzt	“Adresse” expandieren

- Die Beziehungen werden — gemäß Regel 7.4 — wie folgt abgebildet:

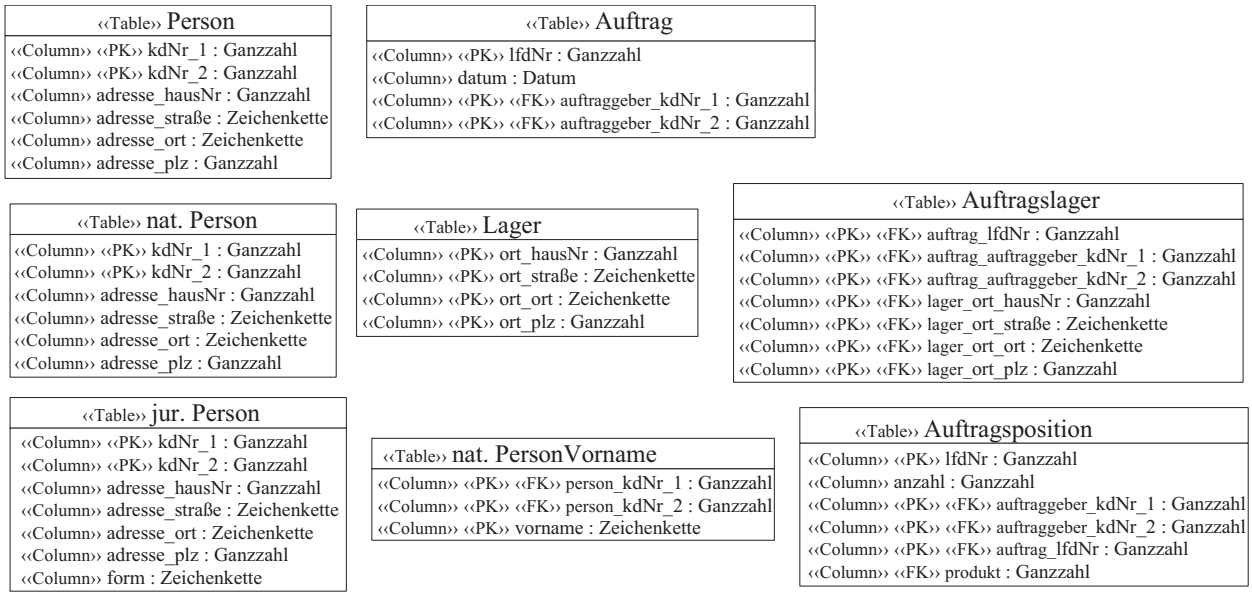
Attribut	**Art**	**Vorgehen**
gibt	1:n	FK bei “Auftrag”
enthält	1:n	FK bei “Auftragsposition”
umfaßt	1:n	Einbettung in “Auftragsposition”
liefert	m:n	separate Tabelle

- Schlüssel sind fachlich wie folgt:

Tabelle	**Schlüssel**
Person	lfdNr
Auftrag	lfdNr, Person::lfdNr
Auftragsposition	lfdNr, Auftrag::lfdNr, Person::lfdNr
Lager	ort

- Die Generalisierung wird nach Variante 2b aufgelöst.

Es ergibt sich dann folgendes Datenbankmodell (konform zu dem Profil in Bild 7.3):



Anmerkung: Eine Bewertung der unterschiedlichen Abbildungen nach diversen Kriterien (wie Platzbedarf, Aufwand für Lesen, usw.) steht in folgender Tabelle. Wie man erkennt, gibt es keine optimale Abbildung, sondern es hängt von der typischen Benutzung ab, welche Variante die beste ist:

Variante / Aufwand für	Denormalisierung	Denormalisierung und Überlagerung	Tabelle mit Objekten des dynamischen Typs	Objekt in allen Vorgängertabellen (alle Attribute)	Objekt in allen Vorgängertabellen (nur Extensionen)
Platz	–	-	++	o	++
Lesen	++	++	++	++	–
polym. Lesen	o	-	–	++	–
Einfügen / Ändern / Löschen	++	++	+	–	-
Änderung im Modell	-	–	++	-	+
Mehrfachvererbung	-	-	++	+	–
komplexe relationale Operationen	++	–	-	+	–

Kapitel 8

Grundlagen des Data Warehousing

8.1 Einführung

8.1.1 Definitionen

Definition 8.1 (Data Warehouse)
Analog zur Definition der Datenbank lässt sich das *Data Warehouse* definieren als eine Sammlung miteinander in Verbindung stehender Daten. In Abgrenzung zu den bisher behandelten Datenbanken ist das Data Warehouse jedoch nicht transaktionsorientiert angelegt, sondern dient dem Zwecke der Entscheidungsunterstützung in der betrieblichen Steuerung. □

In diese Richtung gehen die folgenden Definitionen, die nebeneinander bzw. abhängig vom jeweiligen Kontext verwendet werden.

Definition 8.2 (Data Warehouse nach Bauer und Günzel)
Ein *Data-Warehouse* ist eine physische Datenbank, die eine integrierte Sicht auf (beliebige) Daten darstellt, um Analysen zu ermöglichen. □

Definition 8.3 (Data Warehouse nach Kimball)
A *data warehouse* is a copy of transaction data specifically structured for querying and reporting. □

Definition 8.4 (Data Warehouse nach Zeh)
Ein *Data-Warehouse* ist ein physischer Datenbestand, der eine integrierte Sicht auf die zugrundeliegenden Datenquellen ermöglicht. □

Eine genauere Definition gibt Inmon und liefert damit Kriterien zur Abgrenzung des Data Warehouse gegenüber beliebigen Datenbasen zur Entscheidungsunterstützung.

Definition 8.5 (Data Warehouse nach Inmon)
A *Data-Warehouse* is a subject-oriented, integrated, time-variant, nonvolatile collection of data in support of management´s decision-making process. (Inmon (1996)) □

In hier verwendeten Begriffe werden in der Diskussion um Data Warehouses häufig verwendet werden und seien daher im Folgenden erläutert:

subject-oriented (themenorientiert) Die Auswahl der in das Data-Warehouse zu übernehmenden Daten geschieht nach bestimmten Datenobjekten (Produkt, Kunde, Firma,...), die für die Analysen von Kennzahlen für Entscheidungsprozesse relevant sind, nicht hingegen nach operativen Prozessen oder inneren Abläufen. (z.B. Kunde statt Kundenakquisitaion, Geschäft statt Vertragsabwicklung)

integrated (vereinheitlicht) Im Data-Warehouse werden die in verschiedenen (operativen) Quellsystemen in meist heterogenen Strukturen vorliegenden ausgewählten Daten in vereinheitlichter Form gehalten

time-variant (chronologisiert) Analysen über zeitliche Veränderungen und Entwicklungen sollen im Data Warehouse ermöglicht werden; daher ist die langfristige Speicherung der Daten im Data-Warehouse nötig (Einführung der Dimension "Zeit")

nonvolatile (nicht-flüchtig) Daten werden dauerhaft, also persistent gespeichert.

Definition 8.6 (Data Warehouse nach Inmon, deutsche Fassung aus Wikipedia)
Ein *Data-Warehouse* ist eine themenorientierte, integrierte, chronologisierte und persistente Sammlung von Daten, um das Management bei seinen Entscheidungsprozessen zu unterstützen. □

8.1.2 Abgrenzung gegenüber operativen Systemen (OLTP versus OLAP)

Operative Systeme sind zur Erfüllung konkreter Aufgaben, zur Automatisierung dieser Aufgaben angelegt. Typische Vertreter dafür finden sich z.B. bei

Verwaltungssoftware, Einkaufs- / Verkaufssystemen, Dispositions-, Planungs- und Kontrollsystemen oder allgemein Systemen zur Unterstützung des jeweiligen Geschäftszweckes eines Unternehmens. Die Daten eines solchen Systems sind daher die zu einem laufenden Geschäftsvorfall anfallenden Informationen, die mit der Abarbeitung des Geschäftsvorfalls erstmals erzeugt, geändert und gelesen werden. Dazu ist eine transaktionsorientierte Verarbeitung (OLTP = Online Transactional Processing) mit kurzen Lese- und Schreiboperationen notwendig.

Im Gegensatz dazu erzeugen Data Warehouse Systeme keine eigenen Daten, sie unterstützen keinen Geschäftsvorfall, an dessen Ende Daten zu diesem Geschäftsvorfall vorliegen müssen. Sie fokusieren sich neben der Integration besonders auf die Analysefunktion (OLAP = Online Analytical Processing), wobei durch komplexe Anfragen und lange Lesetransaktionen, Informationen aus den Daten gewonnen werden.

Abgrenzung OLAP- versus OLTP-Konzept:

Charakteristik	OLTP	OLAP
Anwendung	Operatives Geschäft	Analyse und Entscheidungsunterstützung
Nutzer	Sachbearbeiter	Entscheider und Führungskräfte
Datenstruktur	zweidimensional, anwendungsbezogen	multidimensional, themenbezogen
Dateninhalt	detailliert	verdichtete und abgeleitet
Ziel	transaktionale Konsistenzerhaltung	zeitbasierte Versionierung
Aktualität	aktuell	historisch bzw. in Zeitreihen
Frequenz	vom Geschäftsvorfall initiiert	periodisch (Schnappschuss)
Zugriffsform	lesen, schreiben, löschen	verdichten, lesen
Zugriffsmuster	vorhersehbar, repetitiv	ad hoc, heuristisch
-häufigkeit	hoch	mittel bis niedrig
Antwortzeit	kurz (Sekundenbruchteile)	mittel bis lang (Sekunden bis Minuten

8.1.3 Anwendungsbereiche

OLAP-Systeme werden überall da gebraucht, wo auf Basis vorhandener Daten eine Auswertung und Aufbereitung dieser Daten benötigt wird. Die verbreitetsten Anwendungsbereiche dafür sind

CRM(Customer Relationship Management) Kundendaten, die in unterschiedlichen Kontexten gewonnen werden, werden gesammelt und unter (ganz anderen) Gesichtspunkten ausgewertet. Typisch sind an dieser Stelle z.B. Segmentierung und Analyse des Kaufverhaltens, der Kundenbindung etc.

Liquiditätssteuerung Die Verteilung der liquiden Mittel des Unternehmens muss beobachtet und in Relation zu zu erwartendem Liquiditäts-

bedarf gestellt werden.

MIS/EIS (Entscheidungsunterstützung des Managements) Entscheidungen über die zukünftige Ausrichtung eines Unternehmens, über Ausbau oder Einstellen von Geschäftszweigen, Verstärkung von Marktpräsenz, Neuinvestition etc. müssen immer auf Basis "belastbarer" Zahlen getroffen werden.

Bilanzen Meist kein Thema für Data Warehouse Systeme, da es hierfür schon andere Systeme gibt. Jedoch wurden mit Einführung der neuen Rechnungslegungsvorschrift IFRS (International Financial Reporting Standards) in der jüngsten Zeit teilweise die nun schon eingeführten Data Warehouse Konstrukte genutzt. (IFRS sind internationale Rechnungslegungsvorschriften die seit dem 1. Januar 2005 von allen börsennotierten Unternehmen bei der Erstellung ihrer Jahresabschlüsse eingehalten werden müssen. IFRS ist die Fortführung der 1998 eingeführten IAS-Standards (International Accounting Standards).)

Risikosteuerung Risiken für materielle Verluste (z.B. Zahlungsunfähigkeit eines Schuldners, Kursschwankungen) müssen bekannt sein und bzgl. eventueller Häufungen überwacht und ggfls. gegengesteuert werden.

Meldung Ein Thema, das in besonderem Maß Finanzinstitute betrifft, die diversen Meldepflichten gegenüber der Bundesanstalt für Finanzdienstleistungsaufsicht (BaFin) unterworfen sind. Diese Meldung setzt Daten über die Gesamtgeschäfte des Instituts voraus.

8.2 Motivation

8.2.1 Möglichkeiten

Ein Grund für das Entstehen von Analysesystemen war schlichtweg das Vorhandensein der technischen und inhaltlichen Möglichkeiten. Daten fielen im Unternehmen an, wurden zunehmend in EDV-Systemen erfasst bzw. erzeugt und standen damit prinzipiell für jede Nutzung zur Verfügung. Dazu kamen die rapide verbesserten Speicher- und Zugriffsplattformen.

8.2.2 Bedarf

Auf der anderen Seite entstand in den letzten Jahren wachsender Bedarf durch

Marktdruck Sogenannte Decision Support Systeme (DSS) und Executive Information Systeme (EIS) versprachen eine verbesserte Analyse der Gegebenheiten und Entwicklungen und damit eine wesentliche Voraussetzung für qualifizierte Entscheidungen, insbesondere für die Unternehmensstrategie. Damit entsprach man der Notwendigkeit, in schwierigen Zeiten (Globalisierung, wirtschaftliche Schwäche) Wettbewerbsvorteile zu erkennen und nutzen.

Gesetzliche Vorschriften Auch in rechtlichen Regelungen wurde den verbesserten Analysemöglichkeiten Rechnung getragen und insbesondere den Finanzinstituten verschärfte Vorschriften zu Ermittlung bestimmter Kennzahlen und Meldung an BaFin (s.o.), gegeben. Auch die neue Rechnungslegungsvoschrift IFRS lieferte einen weiteren Anwendungsfall.

Rechtfertigungszwang für die IT Die IT der Unternehmen unterliegt einem ständigen Rechtfertigungsdruck wegen der hohen Kosten und des nicht immer für alle transparenten Nutzens der Systeme. Die zusätzlichen Möglichkeiten, bereits vorliegende Daten gewinnbringend einzusetzen verbesserten die Argumentation der IT.

8.3 Architektur

8.3.1 Referenzarchitektur

Die Ausgangssituation für eine Data Warehouse Architektur ist immer dadurch charakterisiert, dass

- Daten aus mehreren Vorsystemen oder externen Datenbeständen
- zusammengeführt und
- zur Nutzung bereitgestellt werden sollen.

Das führt zu einem bestimmten Schema für die benötigte Architektur (siehe Bild 8.1).

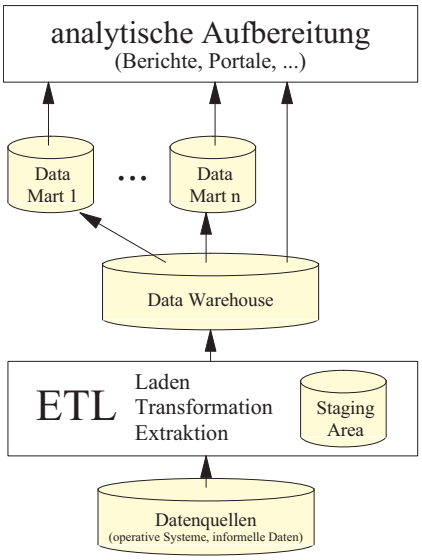


Abbildung 8.1: Referenzarchitektur für Data Warehouses

Dabei ist zunächst offen, in welcher Form die Zusammenführung der Daten erfolgen soll. Jedoch sind dabei die die Anforderungen zu berücksichtigen, die sich aus der gewünschten Analyse ergeben.

Fachliche Anforderungen

- Daten aus den verschiedenen beteiligten operativen Systemen liegen dort in der Regel in unterschiedlicher Form vor, müssen jedoch durch das Data Warehouse *konsolidiert* werden. Dies bezieht sich sowohl auf die Struktur, als auch auf die Befüllung der Daten.

- Für die Analyse müssen die Werte in *Zeitreihen* bereitstehen, in den operativen Systemen gibt es jedoch in der Regel keine Historisierung.

- Bei der Zusammenführung von Datenbeständen muss *Zeitsynchronität* hergestellt werden. Das gilt sowohl für die Daten innerhalb eines Systems, als auch für die Daten unterschiedlicher Systeme. (Transaktion evtl. aus Sicht der Analyse größer gefasst, unterschiedliche Fachliche Zeitpunktdefinition wie z.B. Buchungsschnitt.)

Technische Anforderungen

- Es müssen effiziente *Zugriffsmöglichkeiten* auf die Datenbasis bestehen, um die immensen Datenmengen vernünftig behandlen zu können. Die Zugriffspfade in den operativen Systemen sind jedoch auf deren völlig andere Anwendungsfälle zugeschnitten.

- Die Abläufe in den operativen Systemen dürfen keiner *Beeinträchtigung* ausgesetzt sein, um die Geschäftsabwicklung nicht zu gefährden.

8.3.2 verschiedene Architekturansätze

Es gibt unterschiedliche Ansätze, diese Architektur darzustellen.

- Ein virtuelles Data Warehouse stellt die konsolidierten Daten für die Analyse nur in Form der dazu benötigten Transformationen zur Verfügung. Die Auswertung operiert über die Transformationsvorschriften direkt auf den ursprünglichen Datenbeständen. Es erfolgt keine separate redundante Datenhaltung.

- In einem zentralen Data Warehouse werden Daten aus den operativen Datenquellen und externe Daten zusammengeführt und gemeinsam persistiert. Die so entstandene Datenbasis steht für die Analyse via OLAP-Server zur Verfügung und kann für alle Analyseanwendungen des Unternehmens genutzt werden.

- Dabei kann es allerdings durch die Vielzahl der Anfragen gegen die gleiche Datenbank zu Performanzeinbußen kommen. Deshalb kann es sinnvoll sein, den zentralen Datenbestand gemäß der Zugriffscharakteristiken zu verteilen und gemeinsam als Datenbasis für einen OLAP-Server bereitzustellen. Dieser Ansatz wäre ein verteiltes Data Warehouse.

- Data Marts können einer zentralen oder verteilten Data-Warehouse-Datenbank nachgelagert sein. Sie enthalten Extrakte der Data-Warehouse-Daten. In diesem Fall spricht man auch von abhängigen Data Marts.

- Im Gegensatz dazu sind unabhängige Data Marts nicht oder nur teilweise an die zentrale Datenbasis angeschlossen. Unter Umständen wird aus den Marts das Data Warehouse befüllt. Diese Variante kommt häufig

vor, wenn beim Aufbau des Data-Warehouse-Systems bereits in einzelnen Organisationsbereichen des Unternehmens, unabhängig voneinander eigene kleine Data Warehouses aufgebaut waren und dann für unternehmensweite Analysen integriert wurden.

8.4 Datenstrukturen in OLAP-Systemen

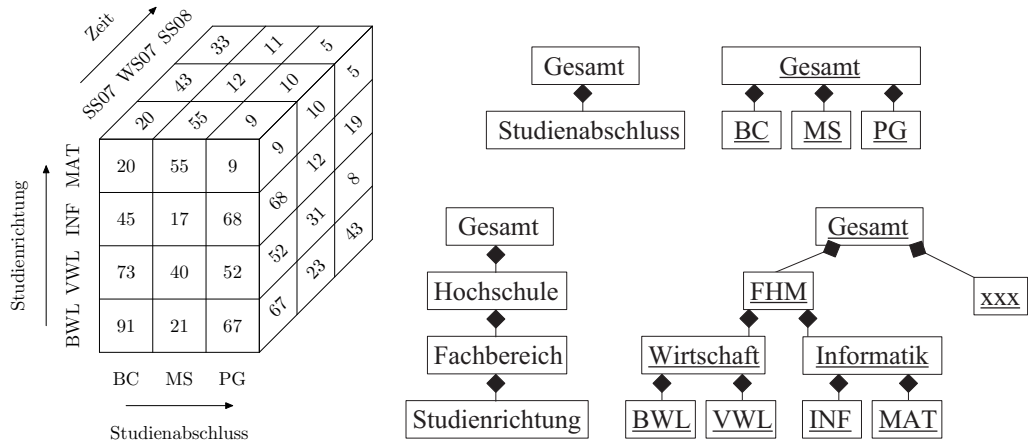
8.4.1 Datenstrukturen

In *OLAP-Systemen* muss, genau wie in OLTP-Systemen, eine Datenstruktur für die zu Grunde liegenden Daten definiert sein. Diese Datenstruktur ähnelt der in operativen Systemen, liegt jedoch auf einer höheren Abtraktionsebene. Diese Daten gehören zu den qualitative Daten.

Das zentrale Objekt von OLAP-Systemen sind aber quantitative Daten, d.h. bestimmte Werteausprägungen von Attributen der analysierten Objekte, meist Zahlen. Diese Werte werden bezüglich verschiedener Kriterien, sozusagen aus verschiedenen Blickwinkeln beleuchtet. Die daraus resultierende Datenstruktur bildet einen mehrdimensionalen Datenraum, den sogenannten Hyperwürfel (Hypercube). Der Würfel wird aufgespannt von den Kriterien, nach denen die Daten beleuchtet werden sollen. Diese Kriterien sind qualitative Daten. Im Inneren des Würfels stehen die quantitativen Daten, die Maßzahlen (Measures).

Begrifflichkeit:

- Eine *Dimension* ist eine konkrete Achse eines Hyperwürfels (z.B. Studienrichtung). Sie bildet einen Aspekt ab.
- Ein *Dimensionselement* ist eine Werteausprägung einer Dimension. Z.B. sind BWL und VWL Dimensionselemente zur Dimension Studienrichtung.
- Die *Dimensionstiefe* ist die Anzahl der Dimensionselemente zu einer Dimension. Die Dimensionstiefe der Dimension Studienrichtung ist bspw. 4.
- Ein *Hyperwürfel* oder *Infotmation Cube* ist ein multidimensionaler Datenraum, der aufgespannt wird von den Dimensionen.
- Die Werte im Inneren des Hyperwürfels heißen *Maßzahlen* (engl. Measures) oder *Kennzahlen*. Dies sind die quantitativen Daten.



(a) (b)

Abbildung 8.2: Hyperwürfel (a) und seine hierarchischen Dimensionen (b)

Bild 8.2 veranschaulicht diese Begriffe.

Man kann einen Hyperwürfel mit zwei Dimensionen als Tabelle abbilden, bei drei Dimensionen erhält man einen Würfel. Ab der vierten Dimension wird die Vorstellung schwierig... Der Hyperwürfel enthält für jede mögliche Kombination von Dimensionswerten eine Maßzahl. Die Anzahl der benötigten Zellen im Hyperwürfel ist daher gleich dem Produkt der Dimensionstiefen aller benötigten Dimensionen.

Häufig weisen die Dimensionselemente in sich eine (hierarchische) Strukturierung auf. Für die Analyse ist dann auch die Auswertung dieser Hierarchie notwendig. Die Auswertung auf einer höheren Hierarchiebene bedeutet dann eine Verdichtung der Maßzahlen. Bspw. können im vorliegenden Beispiel (siehe Bild 8.2) die Zahlen pro Studienrichtung (BWL, VWL, INF, MAT) ausgewertet werden, für die Ebene der Fakultäten (Wirtschaft, Informatik) oder auch für die Hochschule als Ganzes (FH).

8.4.2 Operationen auf Hyperwürfeln

Zur Auswertung der in den Hyperwürfeln liegenden Daten werden Standardoperationen benötigt, die dazu dienen, das Datenmaterial so aufzubereiten, dass der gerade untersuchte Aspekt und ggfls. die darin interessierenden Dimensionselemete in den Vordergrund treten.

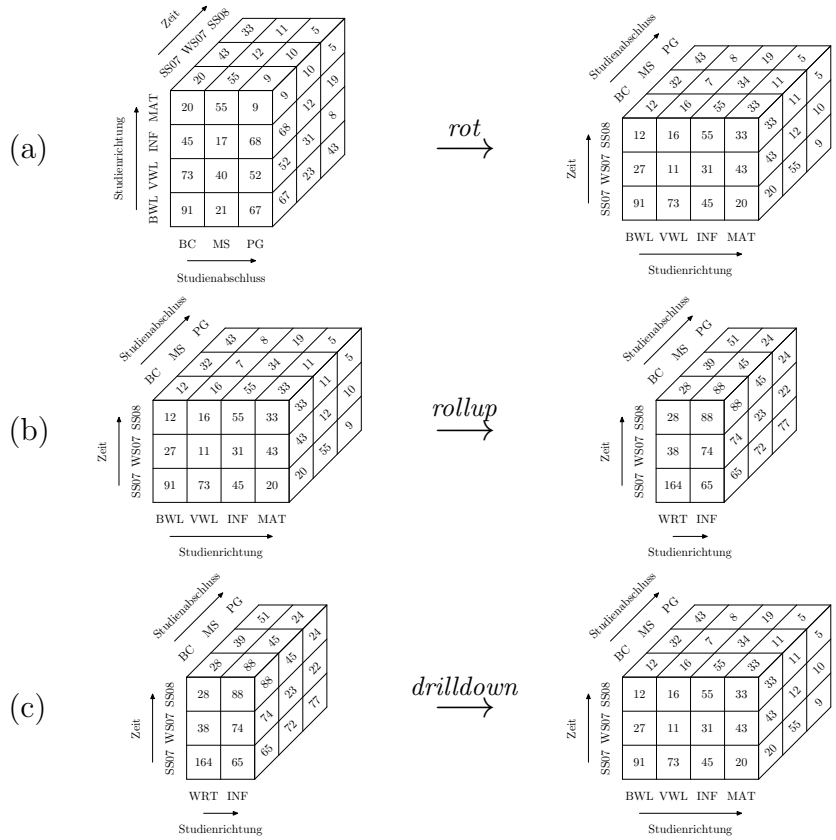


Abbildung 8.3: Operationen auf Würfeln (I): Rotation, Rollup,Drilldown

- Pivotierung/Rotation (Bild 8.3a): Diese Operation dreht den Würfel durch Vertauschen der Dimensionen um seine Achsen. Dadurch ist es möglich, die Daten aus verschiedenen Perspektiven zu analysieren.

- Roll-up, Drill-down (Bilder 8.3b und c): Beim Roll-up werden die Daten entlang des Konsolidierungspfads immer stärker aggregiert. Der Roll-up entspricht also einer Aggregation der Daten. Drill-down ist die zum Roll-up komplementäre Operation. Es erfolgt eine Navigation von verdichteten zu detaillierten Daten.

- Drill-across: Wechsel von einem Hyperwürfel zu einem anderen (z.B. Verkäufe, Umsatz).

- OLAP-Join: Verknüpfung zweier Hyperwürfel zu einem einzigen durch Join entlang gemeinsamer Dimensionen.

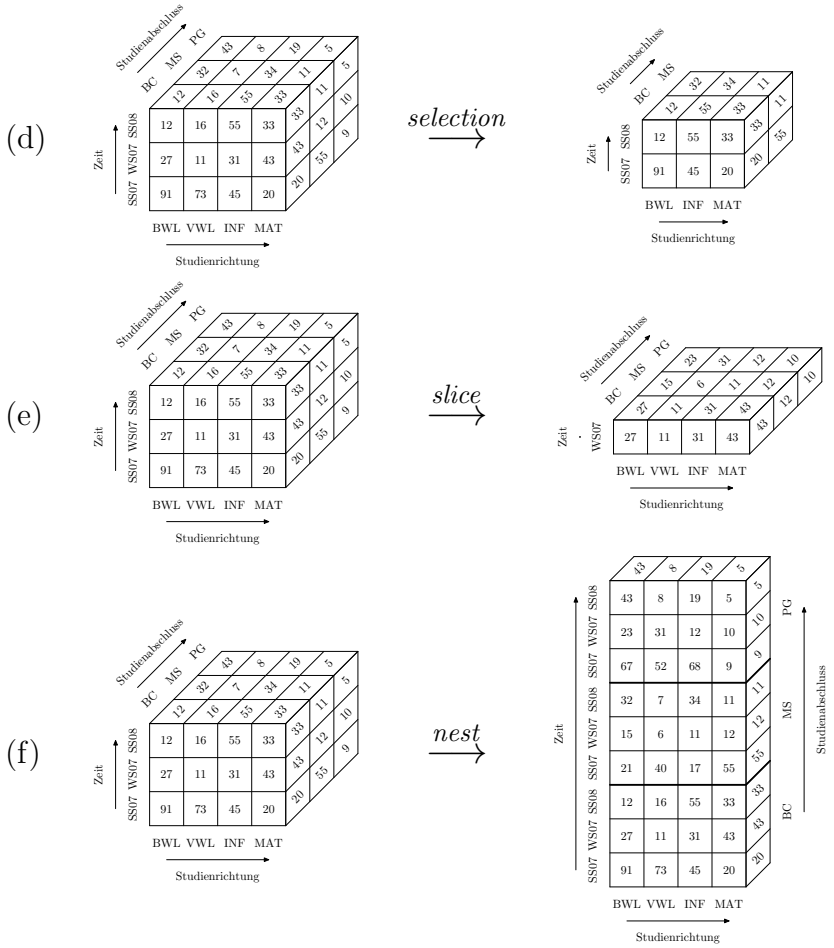


Abbildung 8.4: Operationen auf Würfeln (II): Selektion, Slice, Nest

- Selektion (Bild 8.4d): Es werden Teilwürfel aus dem Hyperwürfel ausgeschnitten, deren Werte bestimmte Bedingungen erfüllen (z.B. die Semester mit den höchsten Studierendenzahlen in der Fakultät Wirtschaft).
- Slice/Dice (Bild 8.4e): Beim Slice werden einzelne Scheiben aus dem Würfel herausgeschnitten, was zu einer Verringerung der Dimensionalität führt. Dice hingegen bezeichnet das Herausschneiden von Teilwürfeln. Beide Operationen sind Sonderfälle der Selektion.
- Nest (Bild 8.4f): Reduktion der Dimensionen durch geschachtelte Unterbringung zweier Dimensionen in einer einzigen. Diese Operation kann durch Zusammenführen mehrere Dimensionen die Anzahl der benötigten Dimensionen auch um mehr als 1 verringern.

8.5 Schritte des Data Warehousing

8.5.1 Extraktion

Die Extraktion ist die Gewinnung der Daten für das Data Warehouse aus den operativen Systemen. Dazu wird in der Regel ein inkrementeller Weg gewählt, d.h. man wird versuchen, Änderungen im Datenbestand zu erkennen und selektiv an das Data Warehouse weiterzugeben. Dazu ist es notwendig, Änderungen zu erkennen und zu protokollieren, so dass auf dieser Information aufbauend die Extraktionder Daten erfolgen kann.

Der Zeitpunkt, zu dem die Extraktion erfolgen soll, kann, abhängig von fachlichen Notwendigkeiten und technischen Möglichkeiten, sehr unterschiedlich festgelegt sein:

- Periodisch: Die Extraktion erfolgt periodisch, wobei die Periodendauer von der Dynamik und den gestellten Anforderungen an die Aktualität der Daten abhängig ist.
- Anfragegesteuert: Eine explizite Anfrage des Datennutzers löst die Extraktion aus.
- Ereignisgesteuert: Der Extraktionsvorgang wird durch ein Zeit-, Datenbank- oder externes Ereignis ausgelöst.
- Real- oder neartime: Bei besonders hohen Anforderungen an die Aktualität der Daten, können Änderungen in den operativen Systemen sofort an das Data-Warehouse-System weitergegeben werden.

Auch in den Fällen, wo keine real- oder neartime-Datenversorgung benötigt wird, besteht aber die Anforderung, nach einem Feststellen der Änderungen, um für die (spätere) Datenübertragung den Umfang der benötigten Lieferung zu kennen. Dazu gibt es verschiedene Möglichkeiten:

- Durch *Monitoring* erfolgt eine permanente Überwachung der Datenquellen hinsichtlich relevanter Datenänderungen. Dieses Monitoring erfolgt in einem eigenen Monitoring Programm. Der Monitor überwacht die Datenquellen hinsichtlich Datenänderungen und hält die gewonnenen Daten für den späteren Datenabzug bereit. Diese wiederum kann sowohl in der bereits erfolgten Zwischenspeicherung der geänderten Daten als auch in den Informationen zum Auffinden der geänderten Datensätze in den Originaldaten des operativen Systems bestehen.

- Sofern in den Datenbanken des operativen Systems ein *Triggger*- Mechanismus unterstützt wird, können an diese Trigger die zur Extraktion oder Extraktionsvorbereitung nötigen Operationen gekoppelt werden.
- Auch auf Basis von *Protokolldateien* (LOGS) der Datenbanksysteme aus den operativen Systemen kann die benötigte Änderungsinformation gewonnen werden.
- Wenn keiner der genannten Mechanismen eingesetzt werden kann, bleibt noch die *Anpassung* der operativen Systeme selbst.

Sollte keine dieser Methoden einsetzbar sein, kann die Änderungserkennung und darauf basierende inkrementelle Datenübertragung auch nachgelagert im Data Warehouse stattfinden.

8.5.2 Transformation

Ziel des Transformationsschrittes ist es, sowohl Daten und Schemata als auch die Datenqualität an die Anwenderanforderungen anzupassen. Die Transformation lässt sich dabei in die Teilschritte Datenbereinigung, Schematransformation und Datentransformation unterteilen.

Bei der *Datenbereinigung* wird versucht, Qualitätsmängel wie beispielsweise fehlerhafte Bezeichnungen oder unsinnige Datumsangaben in den Quelldaten zu beseitigen. Diese Bereinigung muss auf vorgegebene Datenqualitätsmerkmale bzgl. Korrektheit, Konsistenz, Vollständigkeit, Redundanzfreiheit und Einheitlichkeit beschränkt werden. Zu beachten ist, dass Daten nicht relevanter oder aktueller gemacht werden können, als sie sind. Sofern möglich sollte Datenbereinigung bereits in den operativen Systemen selbst stattfinden, insbesondere wenn zur Plausibilisierung domänenspezifisches Wissen benötigt wird.

Bei der *Schematransformation* (bzw. Schemaintegration) werden die Schemata der jeweiligen Datenquelle in ein einheitliches Data-Staging-Schema überführt. Dazu gehören sowohl einfache Formatkonvertierungen als auch komplexe Transformationen, unter Umständen mit anwenderspezifischer Zerlegung oder Komposition einzelner Attribute. Grundsätzlich treten bei der Überführung mehrer heterogener Schemata in ein globales Schema bestimmte Konflikte auf:

Benennungskonflikte z.B. die Verwendung von Synonymen oder Homonymen

Strukturkonflikte z.B. fehlende Attribute an Tabellen,

konfligierende Integritätsbedingungen z.B. unterschiedliche Datentypen oder Wertebereichseinschränkungen bei Attributen

Defaultwert-Konflikte z.B. unterschiedliche Datentypen oder Wertebereichseinschränkungen bei Attributen

Die *Datentransformation* dient dazu, die in der Schematransformation angegebenen Transformationsregeln auf Instanzebene anzuwenden. Im wesentlichen wird dabei zwischen Transformationen auf Datensatzebene und Transformationen auf Attributebene unterschieden). Die Transformation auf Ebene vollständiger Datensätze umfasst dabei die Anwendung einfacher Filterungen sowie Verbundoperationen zur Schemazusammenführung. Auf Attributebene werden hauptsächlich Konvertierungen durchgeführt.

- Erzeugen global eindeutiger Schlüssel
- Anpassung von Datentypen
- Konvertierung von Kodierungen
- Vereinheitlichung von Zeichenketten und Datumsangaben
- Umrechnung von Maÿeinheiten/Skallierung
- Kombination/Separierung von Attributwerten
- Berechnung abgeleiteter Werte und Aggregierung

8.5.3 Laden

Sofern inkrementelle Datenbelieferung erfolgt, gibt es zwei unterschiedliche Ladevorgänge, und zwar die initiale Befüllung und die darauf folgende wiederholte Aktualisierungen des Datenbestands. Wenn im Data-Warehouse-System materialisierte Sichten eingesetzt werden, sind diese bei einer Aktualisierung des Data Warehouse ebenfalls zu aktualisieren. Neben der naiven Methode der kompletten Neuberechung der materialisierten Sicht kann auch der Ansatz der inkrementellen Sichtenwartung verfolgt werden.

8.5.4 Analyse

Nach Bereitstellung der Daten im Data Warehouse kann die Analyse der Daten erfolgen. In dieser werden durch eine Analyse und Interpretation des Datenbestands zweckorientierte Informationen abgeleitet. Hier kommen die bereits eingeführten OLAP Konzepte zum Einsatz. Außerdem kommen in diesem Zusammenhang auch Data Access und Data Mining zum Einsatz, auf die hier jedoch nicht näher eingegangen wird.

Data Access ist die Gewinnung von Informationen aus Daten, auf Grundlage des relationalen Datenmodells, mittels Datenmanipulationssprachen wie beispielsweise SQL. Hierzu kommen häufig Informations-, Berichts- und Planungssysteme zum Einsatz, die standardisierte und flexible Anfragen des Datenbestands ermöglichen. Die Anfragekomplexität dieses Analyseansatzes unterscheidet sich kaum von den Anfragen im OLTP-Bereich.

Online Analytical Processing (OLAP) wird wie beschrieben zur dynamischen, flexiblen und interaktiven Analyse auf eine Vielzahl von Einträgen. Im Unterschied zu Data Access ist das zugrunde liegende Datenmodell an die Analyse angepasst.

Data Mining versucht, Beziehungsmuster in den zugrunde liegenden Daten zu ermitteln und darzustellen. Die so gefundenen Besonderheiten bilden die Basis für weitere Auswertungsschritte wie z.B. Clusterbildung.

8.6 Modellierung

8.6.1 Konzeptionelle Modellierung

Klassische E/R-Modellierung

Prinzipiell könnte man OLAP-Modelle wie klassische Datenmodelle modellieren, d.h. als ein Geflecht von Entitäten und Beziehungen, das möglichst stark normalisiert ist.

Das ist der vom DWH-Guru William Inmon propagierte Ansatz. Er ergänzt solche 3NF-Modelle noch um sogenannte “Subjekte”, die — ähnlich zu UML-Paketen — zusammengehörende Entitäten gruppieren. Beispielsweise könnte

man in einem CRM-System Gruppen wie “Kunde”, “Vertrag”, “Produkt” finden.

Für die Entwicklung solcher Modelle können die für E/R-Modellierung vorgestellten Techniken verwendet werden.

Bei diesem Ansatz braucht man aber oft komplexe Navigationen und Verknüpfungen in Objektnetzen, um zu Daten oder Auswahlen zu kommen. Normale Anwender eines OLAP-Systems sind aber oft Nichtformalisten. Wenn sie eigene Anfragen erstellen können sollen, ist die im wesentlichen tabellarische Sicht eines multidimensionalen Modells einfacher zu handhaben.

Trotzdem kommt ein derartiges Modell oft im operational data store eines Data Warehouses vor.

Multidimensionale Modellierung

Für die fachliche Modellierung von multidimensionalen Würfeln gibt es eigene Notationen, die die Besonderheiten der multidimensionalen Modellierung hervorheben.

Darunter gibt es auch eine Erweiterung der klassischen E/R-Modellierung: die multidimensionale E/R-Modellierung ME/R (von Sapia et al.).

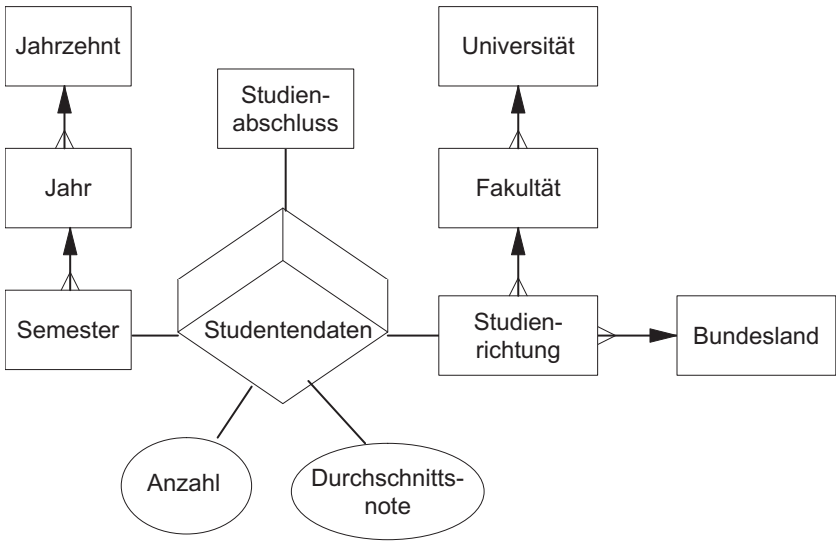
Definition 8.7 (ME/R-Modell)
Ein *ME/R-Modell* besteht aus folgenden Teilen:

- *Dimensionstypen* als Entitäten,
- *Fakten* als n-äre Beziehungen zwischen Dimensionstypen,
- *Kennzahlen* als Attributen von Fakten,
- *Rollup-Beziehungen* als gerichtete Beziehungen zwischen Dimensionstypen.

Die Rollup-Beziehungen bilden einen gerichteten azyklischen Graphen. Damit sind die Dimensionstypen zusammen mit den Rollup-Beziehungen Hierarchien von Klassifikationen für die Fakten. □

Um die ME/R-Modelle von klassischen E/R-Modellen zu unterscheiden, haben die Dimensionstypen, Fakten und Rollup-Beziehungen andere graphische Repräsentationen:

Beispiel 8.8 (ME/R-Modell des Studentenhyperwürfels)
Unser Beispiel des Hyperwürfels würde als ME/R-Modell wie folgt aussehen:



Neben dem vorgestellten gibt es diverse weitere Ansätze (z.B. multidimensional UML, graphbasierte Modelle, ADAPT, ...). Es hat sich aber bisher kein Modellierungsstandard herausgebildet.

8.6.2 Datenbankmodellierung für eine relationale DB

Prinzipiell könnten die Hyperwürfel in entsprechenden multidimensionalen Datenbanken abgelegt werden. Leider sind diese Systeme derzeit noch nicht ausgereift, während sich in den Firmen typischerweise relationale DBMS etabliert haben. Daher bietet sich an, die multidimensionalen Modelle auf relationale umzusetzen ("ROLAP", relational online analytical processing).

Folgende Anforderungen bestehen an die Transformation von einem multidimensionalem Modell in eine relationale DB:

- Multidimensionale Operationen sollen effizient umgesetzt werden können.
- Das relationale Modell soll trotz Redundanz schnell beladen werden können.
- Die Menge an Zieldaten soll nicht zu groß werden.

Für die Ablage der Würfelelemente eines Hyperwürfels in einer relationalen Datenbank sind einige Entwurfsentscheidungen zu treffen.

In jedem Fall gilt aber, dass die Schlüssel der Würfelelementeinträge Fremdschlüssel auf die Dimensionen sein müssen. Die Dimensionsausprägungen sind nämlich die Koordinaten eines Würfelelements im Würfel; daher ist es eindeutig durch sie bestimmt.

Da schnelles Lesen bei einem Würfel im Vordergrund steht und Aktualisierungen praktisch nicht vorkommen, muss das Datenbankmodell eines Hyperwürfels nicht stark normalisiert sein.

Folgende Modellierungsvarianten können gewählt werden.

maximale Denormalisierung Ein Hyperwürfel wird exakt durch eine Tabelle dargestellt. Sämtliche Dimensionsinformation und die Kennzahlen sind in dem Tabelleneintrag eines Würfelelements enthalten. Klassifikationshierarchien werden expandiert und die Dimensionsattribute und fachlichen Schlüssel sind ebenfalls Spalten der Faktentabelle.

Für unser Beispiel würde die Tabelle wie in Bild 8.5 aussehen.

Studentendaten	
≪PK≫	abschlussname : Zeichenkette
≪PK≫	studienrichtungsname : Zeichenkette
≪PK≫	semestername : Zeichenkette
	fakultät : Zeichenkette
	universität : Zeichenkette
	bundesland : Zeichenkette
	jahr : Ganzzahl
	jahrzehnt : Ganzzahl

Abbildung 8.5: Beispiel für maximal denormalisierte Faktentabelle

Vorteile: extrem schneller Zugriff; Dimensionsattribute können deren Stand bei Erstellung wiedergeben

Nachteile: riesige Faktentabellen (in der Regel akademisches Vorgehen); nur **ein** Stand von über die Zeit veränderlichen Dimensionsattributen abzulegen

Sternschema Die Würfelelemente des Hyperwürfels werden durch eine *Faktentabelle* dargestellt, die die Kennzahlen enthält. Schlüssel der Faktentabelle sind Fremdschlüssel auf *Dimensionstabellen*. Diese enthalten

sämtliche Attribute eines Dimensionselements und die Ausprägungen der expandierten Klassifikationshierarchien als Spalten.

Bild 8.6 zeigt unseren Hyperwürfel als Sternschema. Man erkennt die sternförmige Anordnung der Dimensionstabellen um die Faktentabelle.

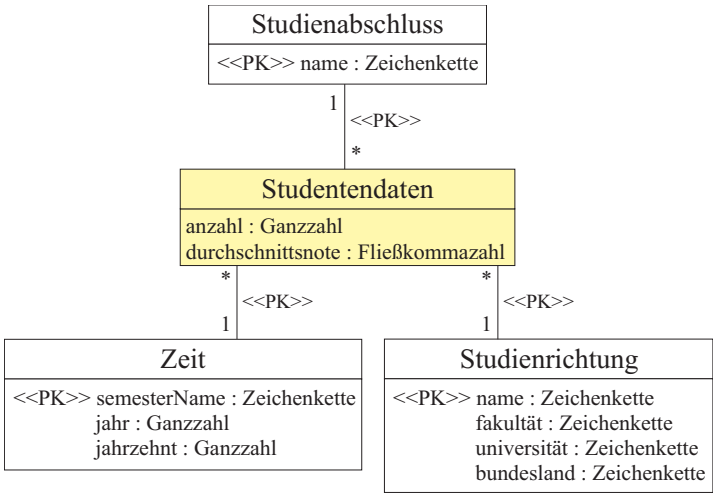


Abbildung 8.6: Beispiel für Sternschema

Vorteile: relativ schneller Zugriff; Faktentabellen mit akzeptabler Größe (enthalten nur noch Fremdschlüssel und Kennzahlen); direkte Dimensionsattribute werden nicht redundant gehalten;

Nachteile: Hierarchien in jeder Dimensionsausprägung vorhanden; intelligente Normalisierung nicht möglich (z.B. bei kulturabhängigen Texten)

Schneeflockenschema Die Würfelelemente des Hyperwürfels werden — wie im Sternschema — durch eine Faktentabelle dargestellt, die die Kennzahlen enthält und Schlüssel der Faktentabelle sind Fremdschlüssel auf Dimensionstabellen. Diese sind normalisiert, d.h. Hierarchien und nicht direkt funktional abhängige Attribute werden in eigene Tabellen abgelegt.

Bild 8.7 zeigt unseren Hyperwürfel als Schneeflockenschema. Der Name kommt daher, dass auch die Dimensionstabellen oft sternförmig Beziehungen zu Nachbartabellen haben.

Vorteile: normalisiert; Fakten- und Dimensionstabellen mit akzeptabler Größe; keine redundante Datenhaltung → klein

Nachteile: Lesezugriffe möglicherweise langsam

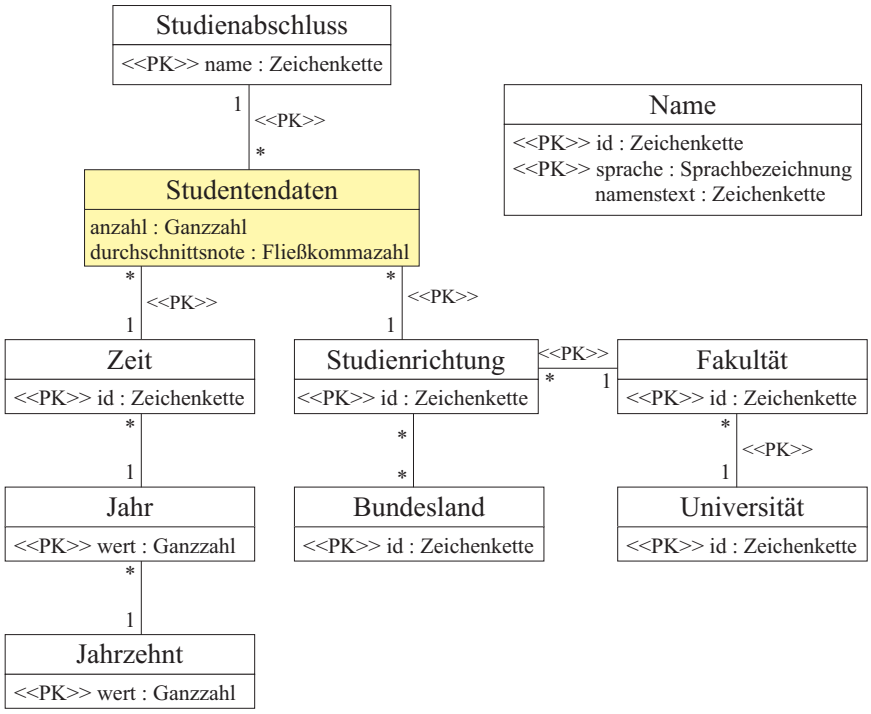


Abbildung 8.7: Beispiel für Schneeflockenchema

8.7 Datenqualität

Vorbemerkung: Der Wert und die Qualität eines Datums sind nicht leicht und nicht allgemein bestimmbar. Ist z.B. 5719920230 richtig, aktuell, wertvoll?

8.7.1 Kriterien

- Interpretierbarkeit,
- Verständlichkeit,
- Vollständigkeit,
- Aktualität,
- Richtigkeit,
- Glaubwürdigkeit,
- Objektivität,

- Relevanz

8.7.2 Anforderungen an Datenlieferanten

Da Datenqualität im Nachhinein nur selten verbessert werden kann, ist es wichtig, bereits an den Datenlieferanten Anforderungen bzgl. der Datenqualität zu stellen. Darüber hinaus kann auch der Datennutzer zu Erkennung von Datenqualitätsproblemen beitragen.

- Richtigkeit der Daten: Sorgfalt bei der Erfassung der Daten, implementieren domänenspezifischer Plausibilitätsprüfungen. Sorgfältige Prüfung der Daten bei der Nutzung.
- Sicherstellung der Aktualität: Zeitnahe Datenerfassung und Prüfen der Aktualität bei der Nutzung.
- Vollständigkeit der Daten: Sorgfältige Erfassung die Daten komplett Nutzung aller relevanten Daten.
- Verfügbarkeit der Daten: Vereinbarungsgemäßes Bereitstellen der Daten für die Data Warehouse Prozesse (zum erforderlichen Zeitpunkt, am erforderlichen Ort und in der erforderlichen Form).
- Übereinstimmung der Daten: Sicherstellen der Datenkonsistenz beim Lieferanten.
- Nutzung der richtigen Liefersysteme
- Aktive Untersuchung und Dokumentation des Datenflusses
- Korrekte Nutzung für DV-Anwendungen absichern: Durch organisatorischen Regelungen, Dokumentationen (Benutzerhandbuch, Betriebskonzepte, Notfallhandbuch, etc.) und Schulungen werden die Nutzer in die Lage versetzt, ein System sachgemnäß zu bedienen.

8.8 Sicherheit

8.8.1 Anforderungen

- Vertraulichkeit (engl. condentiality): Vertraulichkeit bedeutet in diesem Zusammenhang Schutz vor unerlaubtem Zugang zu sensitiven Informationen. Die Vertraulichkeit in einem Data-Warehouse-System muss

sowohl bei direktem Zugriff als auch bei indirekten Anfragen über logische Inferenzen sichergestellt werden.

- Integrität (engl. integrity): Die Anforderung der Integrität zielt auf die Richtigkeit der Daten ab. Es muss sichergestellt sein, dass nur autorisierte Änderungen an Daten vorgenommen werden. Das Ziel der Integrität korrespondiert mit dem Data-Warehouse-Konzept, das eine integrierte, konsistente Sicht auf die Unternehmensbasis darstellt.

- Verfügbarkeit (engl. availability): Die Verfügbarkeit eines Systems hängt davon ab, ob autorisierte Nutzer in denierter Form und innerhalb einer angemessenen Zeit auf die vorhandenen Informationen zugreifen können.

8.8.2 Konzepte zur Sicherstellung der Sicherheitsanforderungen

Benutzeridentikation und Authentisierung Grundlage für ein gesichertes System ist die Erkennung und Verwaltung von Benutzern und Ressourcen, um Sicherheitsbeschränkungen auszuwählen und unauthorisierten Zugriff zu verhindern. Durch entsprechende Mechanismen muss dafür die Authentizität einer vorgegebenen Identität überprüft werden

Auditing Unter Auditing versteht man die Überwachung aller Ereignisse in einem System. Beim Auditing werden zwei Arten unterschieden. Eine Möglichkeit besteht darin, alle Aktionen von allen Benutzern aufzuzeichnen, die zweite Möglichkeit ist die Überwachung im Bezug auf die Verletzung explizit denierter Regeln. Die Auditingmechanismen lassen sich in unterschiedlicher Weise in die Architektur eines Data-Warehouse-Systems einordnen. Im einfachsten Fall können die vorhandenen Auditing-Mechanismen von Datenbankmanagementsystemen ausreichend sein, wenn nur die Tabellenzugrie aufgezeichnet werden sollen. Bei multidimensionalen Anfragen reichen diese Standardmittel meist nicht aus, so dass das Auditing unter Umständen in die multidimensionale Zugrisschicht integriert werden muss

Zugriffskontrolle Während sich die Autorisierung mit der Verwaltung der Zugrisrechte beschäftigt, dient die Zugriskontrolle der Überprüfung dieser Berechtigungen zur Laufzeit. Die Zugriskontrolle baut demnach auf der Autorisierung auf. Die Berechtigungen werden durch eine Autorisierungsinstanz vergeben bzw. entzogen. Wird versucht auf ein

geschütztes Objekt zuzugreifen, werden diese Rechte durch die Zugriskontrollfunktionen geprüft (vgl. [PP04]). Das Hauptaugenmerk der Zugriskontrolle in einem Data-Warehouse-System liegt auf dem Bereich zwischen Data Warehouse und Anwender. Gerade der wachsenden Anwenderkreis mit Zugri auf ein Data Warehouse macht hier ein feingranulares und exibles Zugriskontrollkonzept erforderlich.

Literaturverzeichnis

[1] Andreas Heuer, Gunter Saake.
Datenbanken: Konzepte und Sprachen.
mitp Verlag, Bonn, 2000.

[2] Ramiz ElMasri, Shamkant B. Navathe.
Fundamentals of Database Systems.
Addison-Wesley, Amsterdam, 2003.