Kapitel 1

Einführung

1.1 Definition

Ein **Datenbanksystem** (auch *Datenbankverwaltungssystem*, abgekürzt *DBMS = data base management system*) ist ein computergestütztes System, bestehend aus einer Datenbasis zur Beschreibung eines Ausschnitts der Realwelt sowie Programmen zum geregelten Zugriff auf die Datenbasis.

1.2 Motivation

Die separate Abspeicherung von teilweise miteinander in Beziehung stehenden Daten durch verschiedene Anwendungen würde zu schwerwiegenden Problemen führen:

- **Redundanz:**
  Dieselben Informationen werden doppelt gespeichert.
- **Inkonsistenz:**
  Dieselben Informationen werden in unterschiedlichen Versionen gespeichert.
- **Integritätsverletzung:**
  Die Einhaltung komplexer Integritätsbedingungen fällt schwer.
- **Verknüpfungseinschränkung:**
  Logisch verwandte Daten sind schwer zu verknüpfen, wenn sie in isolierten Dateien liegen.
- **Mehrbenutzerprobleme:**
  Gleichzeitiges Editieren derselben Datei führt zu Anomalien (*lost update*).
- **Verlust von Daten:**
  Außer einem kompletten Backup ist kein Recoverymechanismus vorhanden.
- **Sicherheitsprobleme:**
  Abgestufte Zugriffsrechte können nicht implementiert werden.

- **Hohe Entwicklungskosten:**
  Für jedes Anwendungsprogramm müssen die Fragen zur Dateiverwaltung erneut gelöst werden.

Also bietet sich an, mehreren Anwendungen in jeweils angepaßter Weise den Zugriff auf eine gemeinsame Datenbasis mit Hilfe eines Datenbanksystems zu ermöglichen (Abbildung 1.1).

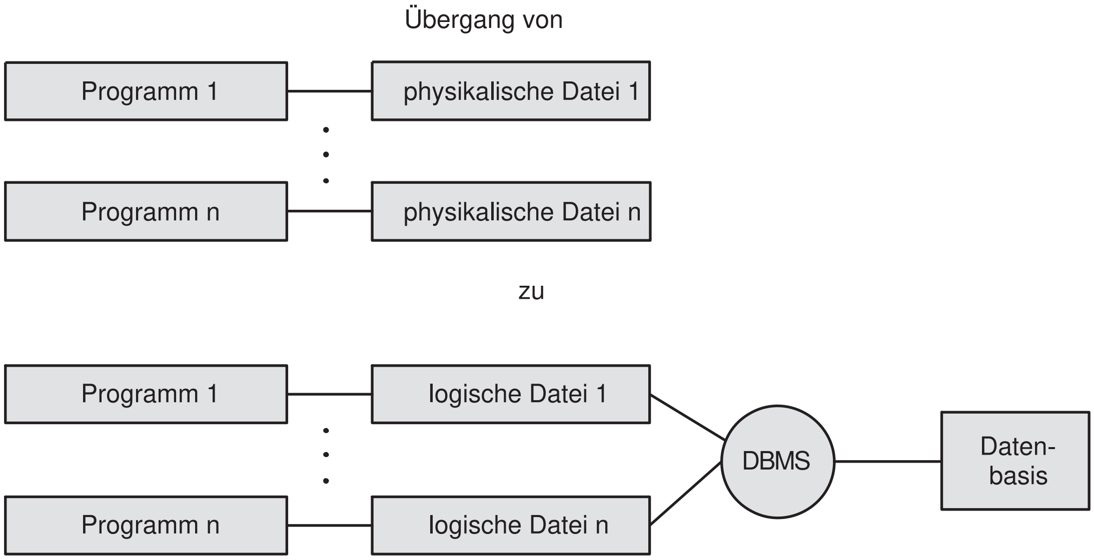


Abbildung 1.1: Isolierte Dateien versus zentrale Datenbasis

1.3 Datenabstraktion

Man unterscheidet drei Abstraktionsebenen im Datenbanksystem (Abbildung 1.2):

- **Konzeptuelle Ebene**
  Hier wird, unabhängig von allen Anwenderprogrammen, die Gesamtheit aller Daten, ihre Strukturierung und ihre Beziehungen untereinander beschrieben. Die Formulierung erfolgt vom *enterprise administrator* mittels einer *DDL (data definition language)*. Das Ergebnis ist das konzeptuelle Schema, auch genannt Datenbankschema.

- **Externe Ebene**
  Hier wird für jede Benutzergruppe eine spezielle anwendungsbezogene Sicht der Daten (*view*) spezifiziert. Die Beschreibung erfolgt durch den *application administrator* mittels einer DDL, der Umgang vom Benutzer erfolgt durch eine *DML (data manipulation language)*. Ergebnis ist das externe Schema.

- **Interne Ebene**
  Hier wird festgelegt, in welcher Form die logisch beschriebenen Daten im Speicher abgelegt werden sollen. Geregelt werden record-Aufbau, Darstellung der Datenbestandteile,

Dateiorganisation, Zugriffspfade. Für einen effizienten Entwurf werden statistische Informationen über die Häufigkeit der Zugriffe benötigt. Die Formulierung erfolgt durch den *database administrator*. Ergebnis ist das interne Schema.

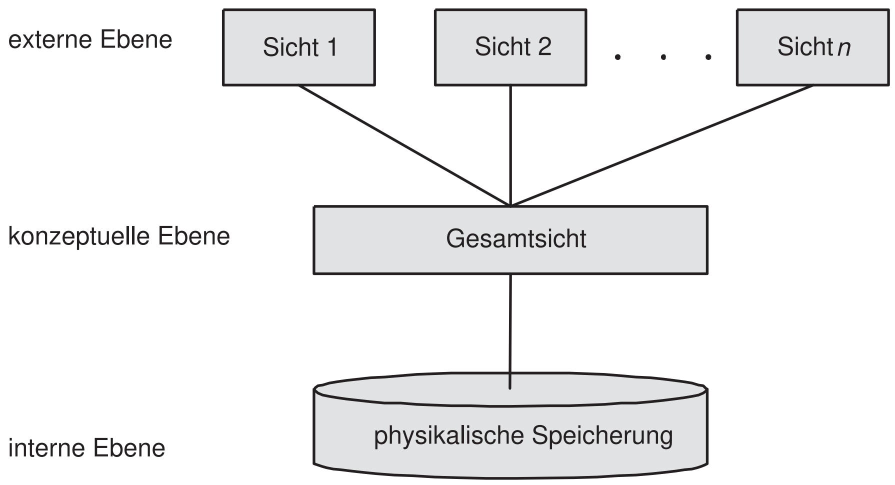


Abbildung 1.2: Drei Abstraktionsebenen eines Datenbanksystems

Das *Datenbankschema* legt also die Struktur der abspeicherbaren Daten fest und sagt noch nichts über die individuellen Daten aus. Unter der *Datenbankausprägung* versteht man den momentan gültigen Zustand der Datenbasis, die natürlich den im Schema festgelegten Strukturbeschreibungen gehorchen muß.

1.4 Transformationsregeln

Die Verbindungen zwischen den drei Ebenen werden durch die *Transformationsregeln* definiert. Sie legen fest, wie die Objekte der verschiedenen Ebenen aufeinander abgebildet werden. Z. B. legt der *Anwendungsadministrator* fest, wie Daten der externen Ebene aus Daten der konzeptuellen Ebene zusammengesetzt werden. Der *Datenbank-Administrator* legt fest, wie Daten der konzeptuellen Ebene aus den abgespeicherten Daten der internen Ebene zu rekonstruieren sind.

- **Beispiel Bundesbahn:**
  Die Gesamtheit der Daten (d. h. Streckennetz mit Zugverbindungen) ist beschrieben im konzeptuellen Schema (Kursbuch). Ein externes Schema ist z. B. beschrieben im Heft *Städteverbindungen Osnabrück*.

- **Beispiel Personaldatei:**
  Die konzeptuelle Ebene bestehe aus Angestellten mit ihren Namen, Wohnorten und Geburtsdaten. Das externe Schema `Geburtstagsliste` besteht aus den Komponenten `Name`, `Datum`, `Alter`, wobei das `Datum` aus Tag und Monat des Geburtsdatums besteht, und `Alter` sich aus der Differenz vom laufenden Jahr und Geburtsjahr berechnet.

Im internen Schema wird festgelegt, daß es eine Datei `PERS` gibt mit je einem record für jeden Angestellten, in der für seinen Wohnort nicht der volle Name, sondern eine Kennziffer gespeichert ist. Eine weitere Datei `ORT` enthält Paare von Kennziffern und Ortsnamen. Diese Speicherorganisation spart Platz, wenn es nur wenige verschiedene Ortsnamen gibt. Sie verlangsamt allerdings den Zugriff auf den Wohnort.

1.5 Datenunabhängigkeit

Die drei Ebenen eines DBMS gewähren einen bestimmten Grad von *Datenunabhängigkeit*:

- **Physische Datenunabhängigkeit:**
  Die Modifikation der physischen Speicherstruktur (z. B. das Anlegen eines Index) verlangt nicht die Änderung der Anwenderprogramme.
- **Logische Datenunabhängigkeit:**
  Die Modifikation der Gesamtsicht (z. B. das Umbenennen von Feldern) verlangt nicht die Änderung der Benutzersichten.

1.6 Modellierungskonzepte

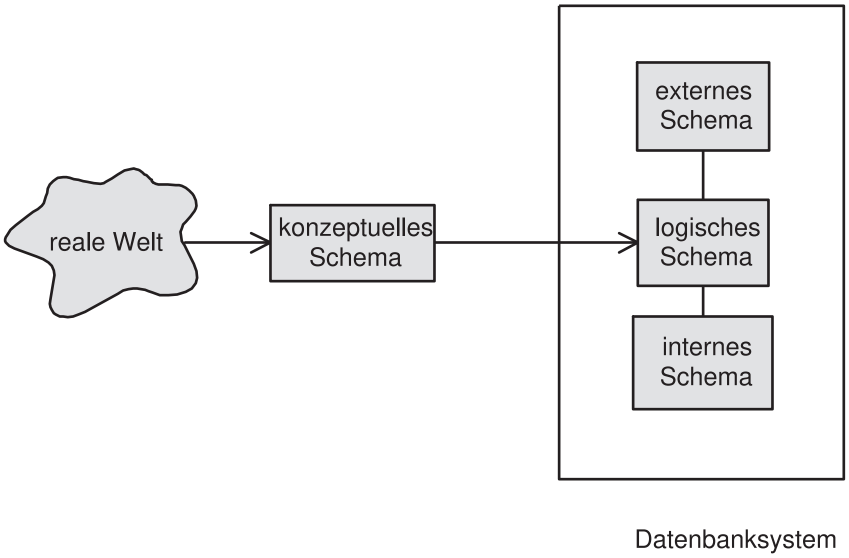


Abbildung 1.3: 2-stufige Modellierung

Das konzeptuelle Schema soll sowohl die reale Welt unabhängig von DV-Gesichtspunkten beschreiben als auch die Grundlage für das interne Schema bilden, welches natürlich stark maschinenabhängig ist. Um diesen Konflikt zu lösen, stellt man dem konzeptuellen Schema ein sogenanntes "logisches" Schema zur Seite, welches die Gesamtheit der Daten zwar hardware-unabhängig, aber doch unter Berücksichtigung von Implementationsgesichtspunkten beschreibt. Das logische Schema heißt darum auch implementiertes konzeptuelles Schema.

Es übernimmt die Rolle des konzeptuellen Schemas, das nun nicht mehr Teil des eigentlichen Datenbanksystems ist, sondern etwas daneben steht und z. B. auch aufgestellt werden kann, wenn überhaupt kein Datenbanksystem zum Einsatz kommt (Abbildung 1.3).

Zur Modellierung der konzeptuellen Ebene verwendet man das **Entity-Relationship-Modell**, welches einen Ausschnitt der Realwelt unter Verwendung von *Entities* und *Relationships* beschreibt :

- **entity:**
  Gegenstand des Denkens und der Anschauung (z. B. eine konkrete Person, ein bestimmter Ort)

- **relationship:**
  Beziehung zwischen den entities (z. B. wohnt in)

Entities werden charakterisiert durch eine Menge von Attributen, die gewisse Attributwerte annehmen können. Entities, die durch dieselbe Menge von Attributen charakterisiert sind, können zu einer Klasse, dem Entity-Typ, zusammengefaßt werden. Entsprechend entstehen Relationship-Typen.

- **Beispiel:**
  Entity-Typ `Student` habe die Attribute `Mat-Nr.`, `Name`, `Hauptfach`.
  Entity-Typ `Ort` habe die Attribute `PLZ`, `Name`.
  Relationship-Typ `wohnt` setzt `Student` und `Ort` in Beziehung zueinander.

Die graphische Darstellung erfolgt durch Entity-Relationship-Diagramme (E-R-Diagramm). Entity-Typen werden durch Rechtecke, Beziehungen durch Rauten und Attribute durch Ovale dargestellt. Abbildung 1.4 zeigt ein Beispiel.

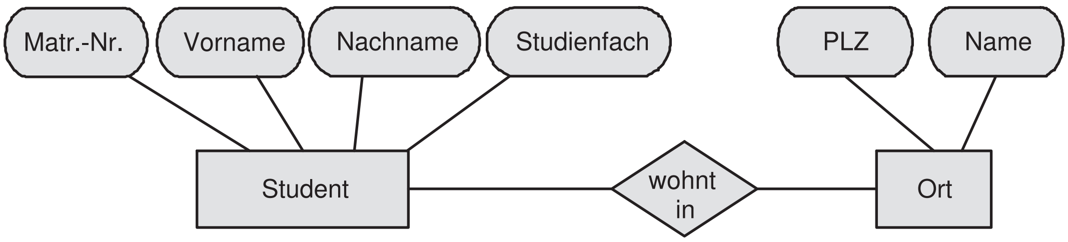


Abbildung 1.4: Beispiel für E-R-Diagramm

Zur Formulierung des logischen Schemas stehen je nach zugrundeliegendem Datenbanksystem folgende Möglichkeiten zur Wahl:

- Das hierarchische Modell (z. B. IMS von IBM)

- Das Netzwerkmodell (z. B. UDS von Siemens)

- Das relationale Modell (z. B. Access von Microsoft)

- Das objektorientierte Modell (z. B. O_2 von O_2 Technology)

Das hierarchische Modell (basierend auf dem Traversieren von Bäumen) und das Netzwerkmodell (basierend auf der Navigation in Graphen) haben heute nur noch historische Bedeutung und verlangen vom Anwender ein vertieftes Verständnis der satzorientierten Speicherstruktur. Relationale Datenbanksysteme (basierend auf der Auswertung von Tabellen) sind inzwischen marktbeherrschend und werden teilweise durch Regel- und Deduktionskomponenten erweitert. Objektorientierte Systeme fassen strukturelle und verhaltensmäßige Komponenten in einem Objekttyp zusammen und gelten als die nächste Generation von Datenbanksystemen.

1.7 Architektur

Abbildung 1.5 zeigt eine vereinfachte Darstellung der Architektur eines Datenbankverwaltungssystems. Im oberen Bereich finden sich vier Benutzerschnittstellen:

- Für häufig zu erledigende und wiederkehrende Aufgaben werden speziell abgestimmte Anwendungsprogramme zur Verfügung gestellt (Beispiel: Flugreservierungssystem).
- Fortgeschrittene Benutzer mit wechselnden Aufgaben formulieren interaktive Anfragen mit einer flexiblen Anfragesprache (wie SQL).
- Anwendungsprogrammierer erstellen komplexe Applikationen durch "Einbettung" von Elementen der Anfragesprache (embedded SQL)
- Der Datenbankadministrator modifiziert das Schema und verwaltet Benutzerkennungen und Zugriffsrechte.

Der DDL-Compiler analysiert die Schemamanipulationen durch den DBA und übersetzt sie in Metadaten.

Der DML-Compiler übersetzt unter Verwendung des externen und konzeptuellen Schemas die Benutzer-Anfrage in eine für den Datenbankmanager verständliche Form. Dieser besorgt die benötigten Teile des internen Schemas und stellt fest, welche physischen Sätze zu lesen sind. Dann fordert er vom Filemanager des Betriebssystems die relevanten Blöcke an und stellt daraus das externe entity zusammen, welches im Anwenderprogramm verarbeitet wird.

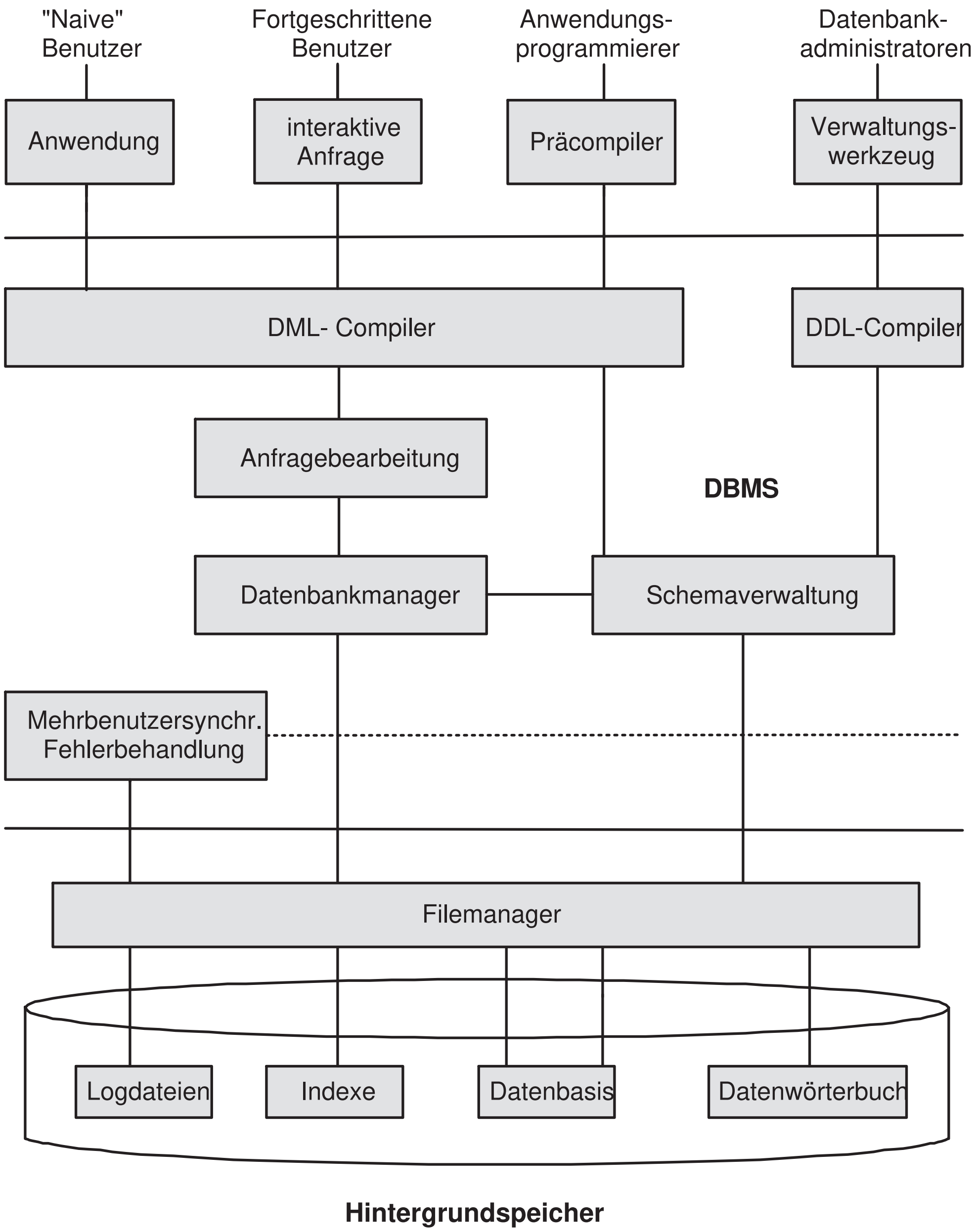


Abbildung 1.5: Architektur eines DBMS

Kapitel 2

Konzeptuelle Modellierung

2.1 Das Entity-Relationship-Modell

Die grundlegenden Modellierungsstrukturen dieses Modells sind die *Entities* (Gegenstände) und die *Relationships* (Beziehungen) zwischen den Entities. Des weiteren gibt es noch *Attribute* und *Rollen*. Die Ausprägungen eines Entity-Typs sind seine Entities, die Ausprägung eines Relationship-Typs sind seine Relationships. Nicht immer ist es erforderlich, diese Unterscheidung aufrecht zu halten.

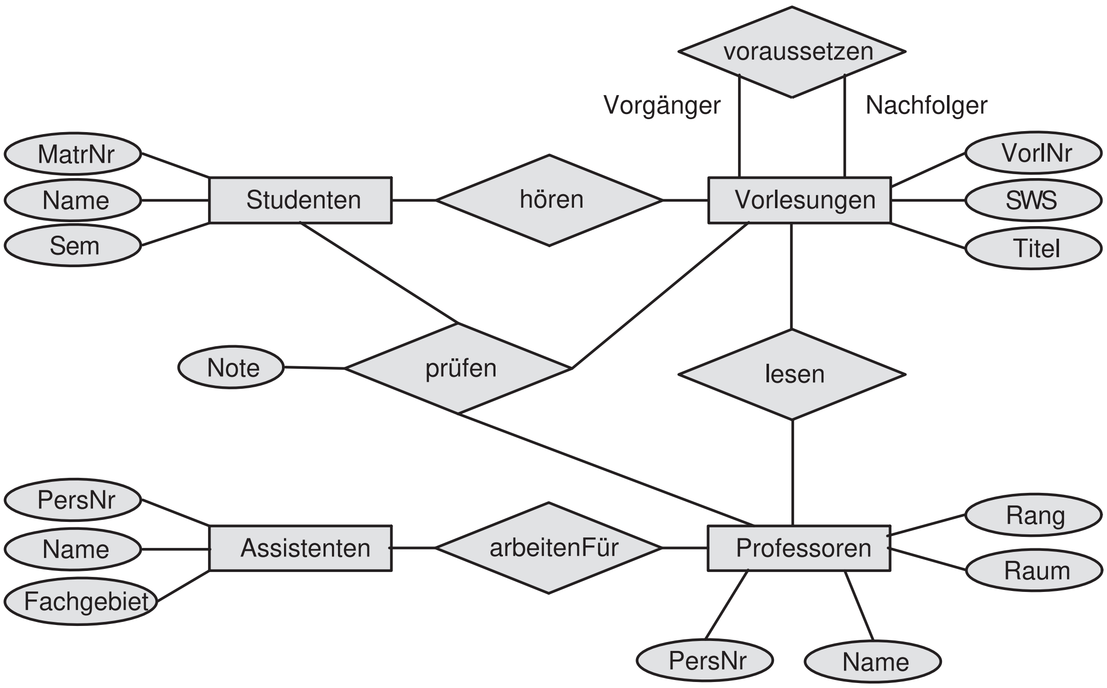


Abbildung 2.1: ER-Diagramm für Universität

Entities sind physisch oder gedanklich existierende Konzepte der zu modellierenden Welt, dargestellt im ER-Diagramm durch Rechtecke. Attribute charakterisieren die Entities und werden durch Ovale beschrieben. Beziehungen zwischen den Entities können binär oder auch mehrwertig sein, sie werden durch Routen symbolisiert.

In Abbildung 2.1 gibt es einen dreiwertigen Beziehungstyp *prüfen*, der auch über ein Attribut *Note* verfügt. Binäre Beziehungstypen, wie z.B. *voraussetzen*, an denen nur ein Entity-Typ beteiligt ist, werden *rekursive Beziehungstypen* genannt. Durch die Angabe von *Vorgänger* und *Nachfolger* wird die Rolle eines Entity-Typen in einer rekursiven Beziehung gekennzeichnet.

2.2 Schlüssel

Eine minimale Menge von Attributen, welche das zugeordnete Entity eindeutig innerhalb aller Entities seines Typs identifiziert, nennt man *Schlüssel*. Gibt es mehrere solcher Schlüsselkandidaten, wird einer als *Primärschlüssel* ausgewählt. Oft gibt es künstlich eingeführte Attribute, wie z.B. Personalnummer *(PersNr)*, die als Primärschlüssel dienen. Schlüsselattribute werden durch Unterstreichung gekennzeichnet. Achtung: Die Schlüsseleigenschaft bezieht sich auf Attribut-Kombinationen, nicht nur auf die momentan vorhandenen Attributwerte!

- **Beispiel:**
  Im Entity-Typ `Person` mit den Attributen `Name`, `Vorname`, `PersNr`, `Geburtsdatum`, `Wohnort` ist `PersNr` der Primärschlüssel. Die Kombination `Name`, `Vorname`, `Geburtsdatum` bildet ebenfalls einen (Sekundär-)Schlüssel, sofern garantiert wird, daß es nicht zwei Personen mit demselben Namen und demselben Geburtsdatum gibt.

2.3 Charakterisierung von Beziehungstypen

Ein Beziehungstyp R zwischen den Entity-Typen E_1, E_2, ..., E_n kann als Relation im mathematischen Sinn aufgefaßt werden. Also gilt:

$$R \subset E_1 \times E_2 \times ... \times E_n$$

In diesem Fall bezeichnet man n als den Grad der Beziehung R. Ein Element $(e_1, e_2, ..., e_n) \in R$ nennt man eine Instanz des Beziehungstyps.

Man kann Beziehungstypen hinsichtlich ihrer *Funktionalität* charakterisieren (Abbildung 2.2). Ein binärer Beziehungstyp R zwischen den Entity-Typen E_1 und E_2 heißt

- *1:1-Beziehung (one-one)*, falls jedem Entity e_1 aus E_1 höchstens ein Entity e_2 aus E_2 zugeordnet ist und umgekehrt jedem Entity e_2 aus E_2 höchstens ein Entity e_1 aus E_1 zugeordnet ist.
  Beispiel: *verheiratet_mit*.

- *1:N-Beziehung (one-many)*, falls jedem Entity e_1 aus E_1 beliebig viele (also keine oder mehrere) Entities aus E_2 zugeordnet sind, aber jedem Entity e_2 aus E_2 höchstens ein Entity e_1 aus E_1 zugeordnet ist.
  Beispiel: *beschäftigen*.

- *N:1-Beziehung (many-one)*, falls analoges zu obigem gilt.
  Beispiel: *beschäftigt_bei*

- *N:M-Beziehung (many-many*, wenn keinerlei Restriktionen gelten, d.h. jedes Entity aus E_1 kann mit beliebig vielen Entities aus E_2 in Beziehung stehen und umgekehrt kann jedes Entity e_2 aus E_2 mit beliebig vielen Entities aus E_1 in Beziehung stehen.
  Beispiel: *befreundet_mit*.

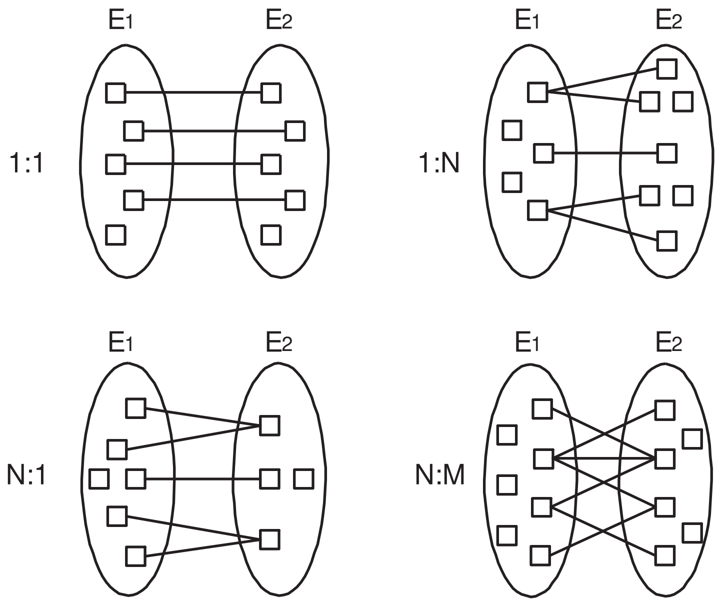


Abbildung 2.2: Mögliche Funktionalitäten von binären Beziehungen

Die binären 1:1-, 1:N- und N:1-Beziehungen kann man auch als *partielle Funktionen* ansehen, welche einige Elemente aus dem Definitionsbereich auf einige Elemente des Wertebereichs abbilden, z. B.

$$\text{beschäftigt\_bei} : \text{Personen} \rightarrow \text{Firmen}$$

2.4 Die (*min, max*)-Notation

Bei der *(min, max)*-Notation wird für jedes an einem Beziehungstyp beteiligte Entity ein Paar von Zahlen, nämlich *min* und *max* angegeben. Dieses Zahlenpaar sagt aus, daß jedes Entity dieses Typs mindestens *min*-mal in der Beziehung steht und höchstens *max*-mal. Wenn es Entities geben darf, die gar nicht an der Beziehung teilnehmen, so wird *min* mit 0 angegeben; wenn ein Entity beliebig oft an der Beziehung teilnehmen darf, so wird die *max*-Angabe durch * ersetzt. Somit ist (0,*) die allgemeinste Aussage. Abbildung 2.3 zeigt die Verwendung der *(min, max)*-Notation anhand der Begrenzungsflächenmodellierung von Polyedern.

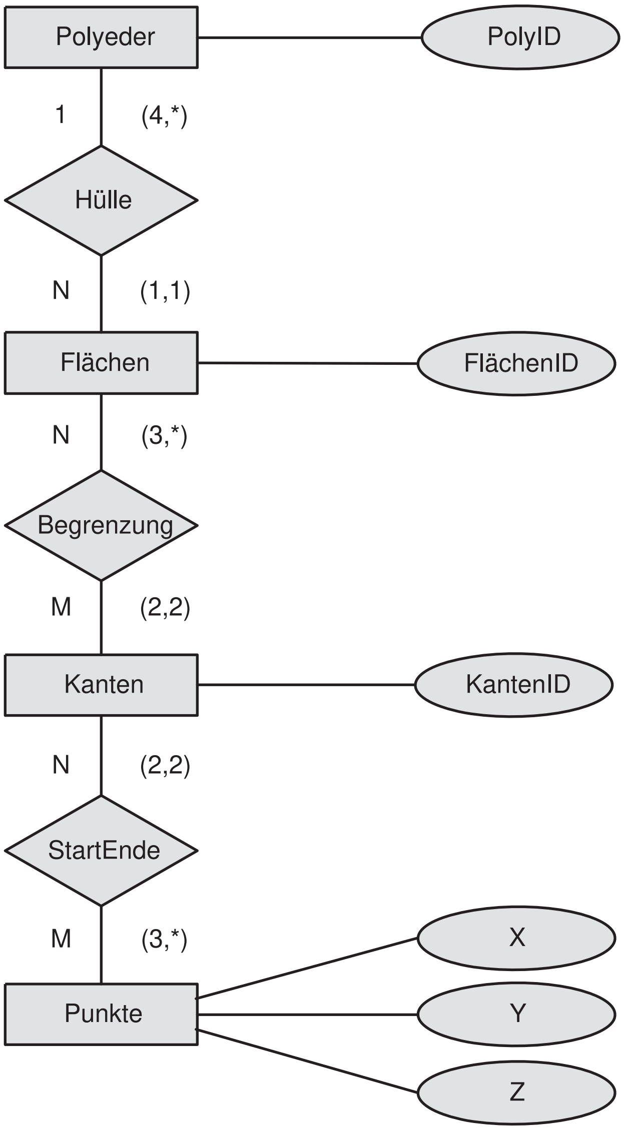


Abbildung 2.3: ER-Diagramm für Begrenzungsflächendarstellung von Polyedern

2.5 Existenzabhängige Entity-Typen

Sogenannte *schwache* Entities können nicht autonom existieren, sondern

- sind in ihrer Existenz von einem anderen, übergeordneten Entity abhängig
- und sind nur in Kombination mit dem Schlüssel des übergeordneten Entity eindeutig identifizierbar.

Abbildung 2.4 verdeutlicht dieses Konzept anhand von Gebäuden und Räumen. Räume können ohne Gebäude nicht existieren. Die Raumnummern sind nur innerhalb eines Gebäudes eindeutig. Daher wird das entsprechende Attribut gestrichelt markiert. Schwache Entities werden durch doppelt gerahmte Rechtecke repräsentiert und ihre Beziehung zum übergeordneten Entity-Typ durch eine Verdoppelung der Raute und der von dieser Raute zum schwachen Entity-Typ ausgehenden Kante markiert.

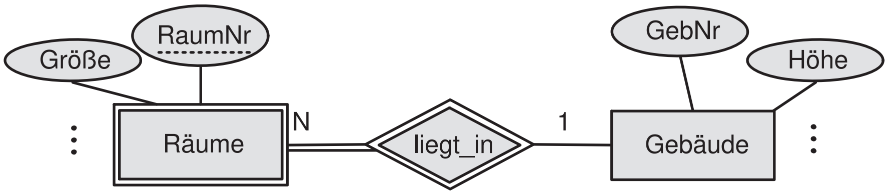


Abbildung 2.4: Ein existenzabhängiger (schwacher) Entity-Typ

2.6 Generalisierung

Zur weiteren Strukturierung der Entity-Typen wird die *Generalisierung* eingesetzt. Hierbei werden Eigenschaften von ähnlichen Entity-Typen einem gemeinsamen *Obertyp* zugeordnet. Bei dem jeweiligen *Untertyp* verbleiben nur die nicht faktorisierbaren Attribute. Somit stellt der Untertyp eine *Spezialisierung* des Obertyps dar. Diese Tatsache wird durch eine Beziehung mit dem Namen **is-a** (ist ein) ausgedrückt, welche durch ein Sechseck, verbunden mit gerichteten Pfeilen symbolisiert wird.

In Abbildung 2.5 sind *Assistenten* und *Professoren* jeweils Spezialisierungen von *Angestellte* und stehen daher zu diesem Entity-Typ in einer *is-a* Beziehung.

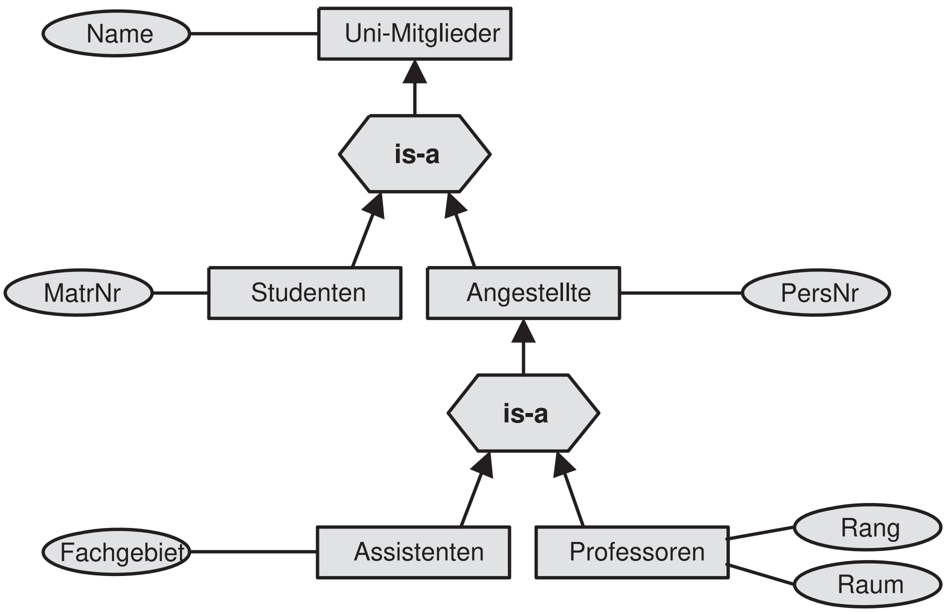


Abbildung 2.5: Spezialisierung der Universitätsmitglieder

Bezüglich der Teilmengensicht ist von Interesse:

- die *disjunkte* Spezialisierung: die Entitymengen der Untertypen sind paarweise disjunkt
- die *vollständige* Spezialisierung: die Obermenge enthält keine direkten Elemente, sondern setzt sich komplett aus der Vereinigung der Entitymengen der Untertypen zusammen.

In Abbildung 2.5 ist die Spezialisierung von *Uni-Mitglieder* vollständig und disjunkt, die Spezialisierung von *Angestellte* ist disjunkt, aber nicht vollständig, da es noch andere, nichtwissenschaftliche Angestellte (z.B. Sekretärinnen) gibt.

2.7 Aggregation

Durch die *Aggregation* werden einem übergeordneten Entity-Typ mehrere untergeordnete Entity-Typen zugeordnet. Diese Beziehung wird als *part-of* (Teil von) bezeichnet, um zu betonen, daß die untergeordneten Entities Bestandteile der übergeordneten Entities sind. Um eine Verwechselung mit dem Konzept der Generalisierung zu vermeiden, verwendet man nicht die Begriffe *Obertyp* und *Untertyp*.

Abbildung 2.6 zeigt die Aggregationshierarchie eines Fahrrads. Zum Beispiel sind *Rohre* und *Lenker* Bestandteile des *Rahmen*; *Felgen* und *Speichen* sind Bestandteile der *Räder*.

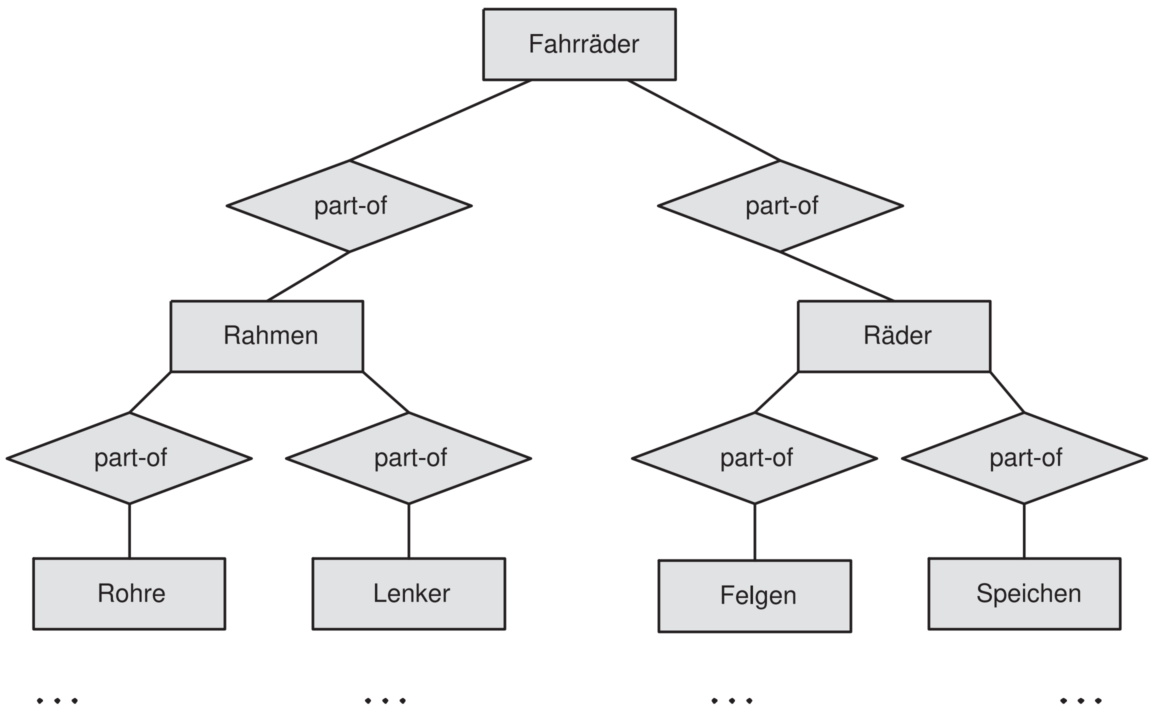


Abbildung 2.6: Aggregationshierarchie eines Fahrrads

2.8 Konsolidierung

Bei der Modellierung eines komplexeren Sachverhaltes bietet es sich an, den konzeptuellen Entwurf zunächst in verschiedene Anwendersichten aufzuteilen. Nachdem die einzelnen Sichten modelliert sind, müssen sie zu einem globalen Schema zusammengefaßt werden (Abbildung 2.7).

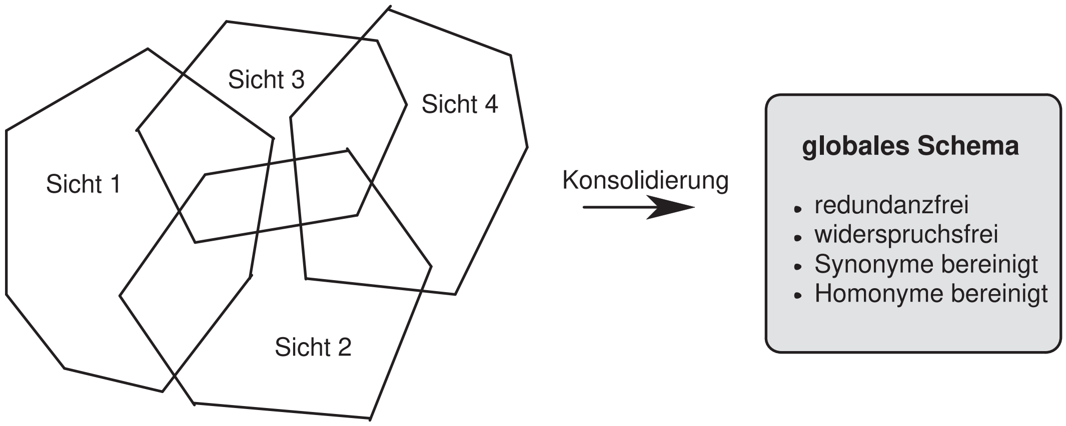


Abbildung 2.7: Konsolidierung überlappender Sichten

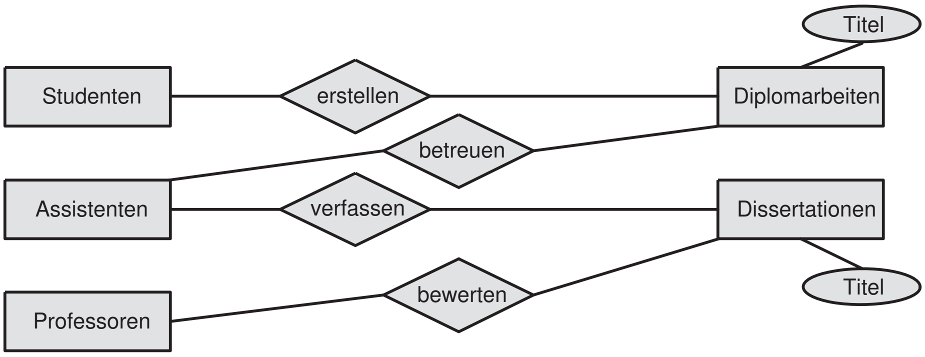
Probleme entstehen dadurch, daß sich die Datenbestände der verschiedenen Anwender teilweise überlappen. Daher reicht es nicht, die einzelnen konzeptuellen Schemata zu vereinen, sondern sie müssen *konsolidiert* werden.

Darunter versteht man das Entfernen von Redundanzen und Widersprüchen. Widersprüche entstehen durch *Synonyme* (gleiche Sachverhalte wurden unterschiedlich benannt) und durch *Homonyme* (unterschiedliche Sachverhalte wurden gleich benannt) sowie durch unterschiedliches Modellieren desselben Sachverhalts zum einen über Beziehungen, zum anderen über Attribute. Bei der Zusammenfassung von ähnlichen Entity-Typen zu einem Obertyp bietet sich die Generalisierung an.

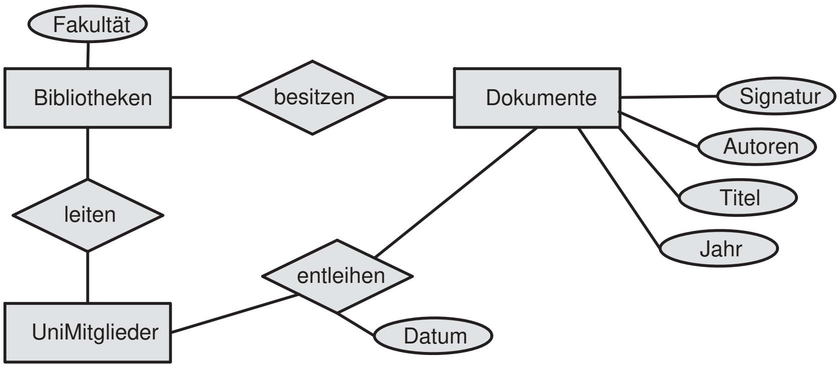
Abbildung 2.8 zeigt drei Sichten einer Universitätsdatenbank. Für eine Konsolidierung sind folgende Beobachtungen wichtig:

- *Professoren* und *Dozenten* werden synonym verwendet.
- *UniMitglieder* ist eine Generalisierung von *Studenten*, *Professoren* und *Assistenten*.
- *Bibliotheken* werden nicht von beliebigen *UniMitglieder* geleitet, sondern nur von *Angestellte*.
- *Dissertationen*, *Diplomarbeiten* und *Bücher* sind Spezialisierungen von *Dokumente*.
- Die Beziehungen *erstellen* und *verfassen* modellieren denselben Sachverhalt wie das Attribut *Autor*.

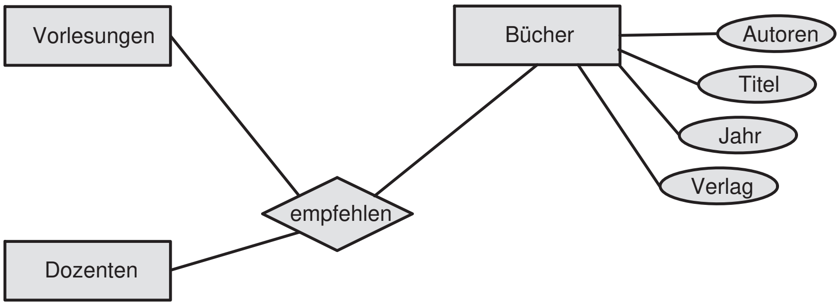
Abbildung 2.9 zeigt das Ergebnis der Konsolidierung. Generalisierungen sind zur Vereinfachung als fettgedruckte Pfeile dargestellt. Das ehemalige Attribut *Autor* ist nun als Beziehung zwischen Dokumenten und Personen modelliert. Zu diesem Zweck ist ein neuer Entity-Typ *Personen* erforderlich, der *UniMitglieder* generalisiert. Damit werden die ehemaligen Beziehungen *erstellen* und *verfassen* redundant. Allerdings geht im konsolidierten Schema verloren, daß Diplomarbeiten von *Studenten* und *Dissertationen* von *Assistenten* geschrieben werden.



Sicht 1: Erstellung von Dokumenten als Prüfungsleistung



Sicht 2: Bibliotheksverwaltung



Sicht 3: Buchempfehlungen für Vorlesungen

Abbildung 2.8: Drei Sichten einer Universitätsdatenbank

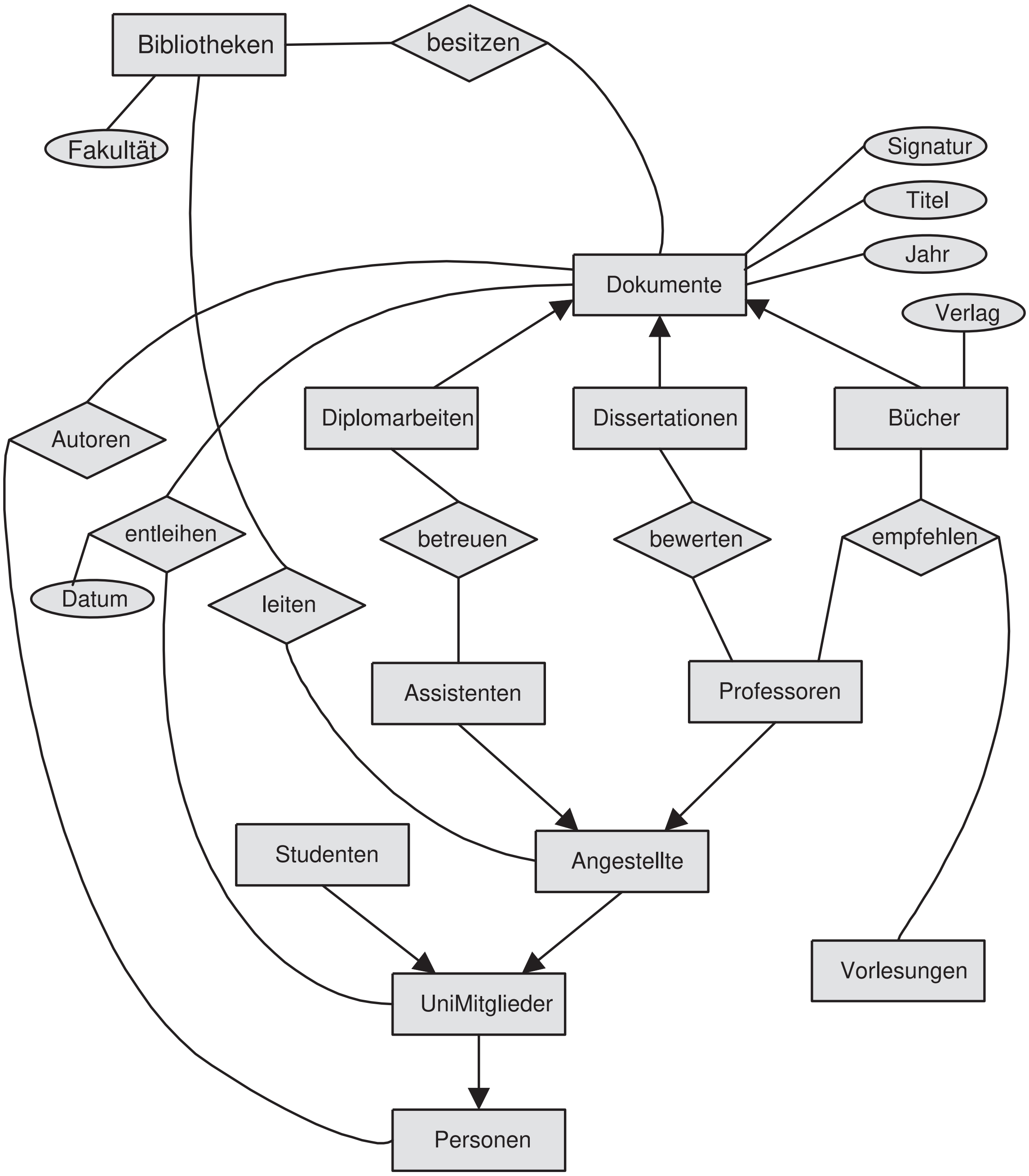


Abbildung 2.9: Konsolidiertes Schema der Universitätsdatenbank