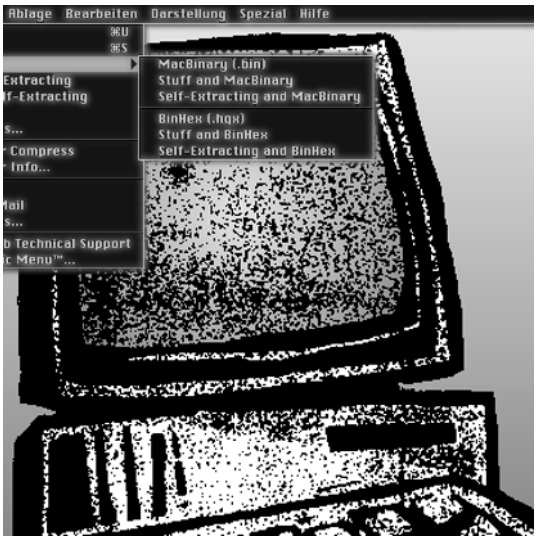
Grundlagen der Wirtschaftsinformatik

Lerneinheit 2: Datenbanken



Prof. Dr. Peter Hartel

004505-001804/Dezember 2019

VERBUNDSTUDIUM
Berufsbegleitend zum Hochschulabschluss

Diese Lerneinheit wurde für den Studiengang Betriebswirtschaft (B.A) der Fachhochschule Bielefeld und der Hochschule Bochum entwickelt und wird im Verbundstudium der Fachhochschulen Nordrhein-Westfalens eingesetzt.

Stand: Dezember 2019

Redaktion: Institut für Verbundstudien der Fachhochschulen Nordrhein-Westfalens – IfV NRW
Im Alten Holz 131, D-58093 Hagen
Telefon +49 (0) 2331 9330–901, Telefax +49 (0) 2331 9330–903
Internet: http://www.ifv-nrw.de/

Inhaltsverzeichnis

Vorwort

Informationen werden in modernen Unternehmen als relevante Produktionsfaktoren erkannt. Damit werden die Organisation der Informationsverarbeitung und das Datenmanagement zu einer zentralen Aufgabe. Ein funktionierendes Datenmanagement setzt angemessene, d.h. an die betrieblichen Abläufe angepasste **Informationssysteme** voraus. Das Speichern und Bereitstellen von Daten erfolgt dabei über **Datenbanken**, die das zentrale Thema dieser Lerneinheit sind.

Die vorliegende Lerneinheit gliedert sich in sieben Kapitel. Das erste Kapitel ordnet das Thema der Datenbanken in den allgemeinen Kontext der Informationsverarbeitung ein. In dem zweiten Kapitel werden allgemeine Grundlagen von Datenbanksystemen erläutert. Der konzeptuelle Datenbankentwurf, in dem Datenbankanforderungen aus einer anwenderorientierten Sicht formal dokumentiert werden, wird im Kapitel 3 eingeführt. Der logische Datenbankentwurf, der die Basis für die Implementierung der Datenbank bildet, steht im Mittelpunkt der Kapitel 4 und 5. Einige wesentliche Aspekte der Datenbanksprache SQL werden in dem Kapitel 6 vorgestellt.

Nach dem Durcharbeiten dieser Lerneinheit sollten Sie:

- Die Begriffe Datenbank und Datenbanksysteme definieren können.
- Aktuelle Entwicklungen aus dem Bereich Big Data benennen können.
- Die Phasen des Datenbankentwurfs erläutern können und benennen können, welche Aufgaben mit welchen Hilfsmitteln und welchen Zielen in den einzelnen Phasen zu erledigen sind.
- Die Konzepte des ER-Modells kennen und in der Lage sein, einen konzeptuellen Datenbankentwurf zu erstellen.
- Das Relationen Modell kennen und ein konzeptuelle Datenbankschema in ein logisches Datenbankschema transformieren können.
- In der Lage sein, Daten aus komplex strukturierte Datenbanken beliebiger Größe abzufragen.

Jedes Kapitel enthält eine Vielzahl von Übungsaufgaben, deren Bearbeitung einem tieferen Verständnis der vorgestellten Konzepte dienen. Musterlösungen zu den meisten dieser Aufgaben sind im Anhang (Kapitel 7) zu finden.

Trotz allen Bemühens ist es nicht auszuschließen, dass sich an der einen oder anderen Stelle ein Fehler in einem Beispiel oder einer Abbildung eingeschlichen hat. Daher übernimmt für die in dieser Lerneinheit gemachten Ausführungen und vorgestellten Verfahren in Bezug auf die fehlerfreie Funktion und deren Wirtschaftlichkeit weder das Institut für Verbundstudien noch der Autor irgendeine Garantie oder die juristische Haftung.

In dieser Lerneinheit werden Hardware- und Softwarebezeichnung verwendet, die eingetragene Warenzeichen sind und auch ohne besondere Kennzeichnung als solche zu betrachten sind.

1 Datenbanken und Informationssysteme

Die Bedeutung die Informationen und deren Verarbeitung mittels Informations- und Kommunikationssysteme in modernen Gesellschaften wird durch den Begriff der **Informationsgesellschaft** charakterisiert. Die Verfügbarkeit von Informationen spielt in nahezu allen Lebensbereichen eine immer wichtiger werdende Rolle. Aus der Sicht der Informationstechnologie sind dabei Datenbanken bzw. Datenbanksysteme von zentraler Bedeutung, da sie die digitale Speicherung und Verarbeitung von beliebig großen Datenmengen ermöglichen.

Beobachtet man aktuelle Entwicklungen in Wirtschaft und Gesellschaft, stellt man fest, dass Daten zu einem beherrschenden Thema für zukünftige Entwicklungen werden. Schlagworte wie Big Data und künstliche Intelligenz prägen die Diskussion und versprechen große Potenziale in nahezu allen Lebens- und Wirtschaftsbereichen. Nicht umsonst werden *Daten* daher als der Rohstoff oder das Erdöl der Zukunft bezeichnet.

Die meisten Menschen kommen in ihrem täglichen Leben mit Datenbanken in Berührung, z.B. wenn sie Geld an einem Bankautomaten abheben, eine Bestellung im Internet tätigen, Informationen mit einer Suchmaschine recherchieren oder in einem Supermarkt einkaufen. Bei diesen Aktivitäten wird auf Datenbanken zugegriffen, um Informationen zu suchen, zu speichern oder zu aktualisieren. Allerdings bleibt bei diesen Vorgängen die Datenbank im Verborgenen. Der Zugriff erfolgt im Prinzip immer über den Umweg eines Informationssystems, das dem Anwender eine mehr oder weniger komfortable Benutzeroberfläche bietet.

Der Begriff **Informationssysteme** sollte aus der ersten Lerneinheit bereits bekannt sein. Zusammenfassend sind Informationssysteme Softwaresysteme, die zur Erfassung, Verarbeitung, Speicherung, Auswertung und Anzeige von Informationen dienen. Datenbanken sind ein integraler Bestandteil jedes Informationssystems und haben die Aufgabe, Daten dauerhaft und sicher zu speichern. Informationssysteme ohne Datenbanken sind im Prinzip nicht denkbar!

Informationssysteme, die in Organisationen und Unternehmen zum Einsatz kommen, werden auch als betriebliche Informationssysteme bezeichnet. Sie steuern Produktions- und Fertigungsprozesse, sie erlauben es Kunden, im Internet Waren einzukaufen und zu bezahlen, sie analysieren große Datenmengen für Entscheidungsprozesse, und dokumentieren betriebliche Abläufe und Vorgänge.

Die Speicherung und Manipulation großer Datenmengen ist eine zentrale Aufgabe von Informationssystemen und erfolgt mittels Datenbanken, die eine dauerhafte Speicherung von Daten erlauben. Aus einer konzeptionellen Sicht kann der Datenbankbegriff wie folgt definiert werden:

*Eine **Datenbank** ist eine Sammlung von logisch zusammenhängenden Daten, die einen Ausschnitt der realen Welt beschreiben und die von einer Gruppe von Benutzern für einen bestimmten Zweck eingesetzt wird.*

2 Grundlagen von Datenbanken

Für den Anwender präsentiert sich der Computer über ein Betriebssystem. Das Betriebssystem verwaltet die Betriebsmittel[1] des Computers und stellt dem Anwender eine Reihe grundlegender Funktionen zur Verfügung. Zur Speicherung von Daten kennt das Betriebssystem das Konzept der **Datei**. Dateien lassen sich dauerhaft auf den unterschiedlichsten Speichermedien ablegen. Das Betriebssystem verwaltet die Dateien und erlaubt es, neue Dateien anzulegen, bestehende Dateien zu löschen, Daten aus Dateien zu lesen oder in Dateien zu schreiben.

Die meisten Computerprogramme (z.B. Office-Anwendungen oder CAD-Programme) verwenden Dateien zur Speicherung von Daten. Daher ist es naheliegend, Dateien auch zur Massendatenspeicherung und somit als Datenbanken einzusetzen. Das würde bedeuten, dass jedes Informationssystem über eine eigene Dateiorganisation verfügt, die auf die speziellen Anforderungen des jeweiligen Informationssystems ausgerichtet ist. Das hätte unter anderem zur Folge, dass

- genau die Daten gespeichert werden können, die für die spezielle Anwendung von Interessen sind,
- die Datenstrukturen und die Datencodierung auf die besonderen Anforderungen der Anwendung und die Möglichkeiten des verwendeten Computersystems hin optimiert werden.

Ein solches Vorgehen würde also eine auf das einzelne Informationssystem ausgerichtete optimale Datenspeicherung ermöglichen. Allerdings ergibt sich eine Vielzahl massiver Nachteile, die gegen die Verwendung eines **Dateiansatzes**[2] sprechen:

- bei der Verwendung von Dateien werden Daten häufig getrennt voneinander und mehrfach gespeichert. Diese Mehrfachspeicherung bezeichnet man auch als **Redundanz**. Redundanzen bringen die Gefahr von **Inkonsistenzen** mit sich, die entstehen, wenn bei Änderungen eines mehrfach gespeicherten Informationsobjektes, diese Änderungen nicht synchron an allen Speicherorten vorgenommen werden. Die systematische Überwachung von Redundanzen bei der Verwendung eines Dateiansatzes ist ein organisatorisch und technisch kaum lösbares Problem.
- Die Verwendung unterschiedlicher Datenstrukturen macht es nahezu unmöglich, Daten, die von verschiedenen Anwendungen gespeichert wurden, integriert auszuwerten[3].
- Der unternehmensweite Einsatz von Informationssystemen setzt einen **Mehrbenutzerbetrieb** voraus. D.h. beliebig viele Anwender

1 Unter den Betriebsmitteln eines Computers versteht man die Systemelemente, die zur Ausführung eines Programms benötigt werden. Dazu gehören z.B. die CPU, der Hauptspeicher und die Festplatte.

2 Der Dateiansatz ist durch die zuvor beschriebene Massendatenspeicherung in Betriebssystemdateien gekennzeichnet.

3 Z.B. benötigt ein PPS-System für die Produktionsplanung aktuelle Bestandsdaten aus dem Logistiksystem

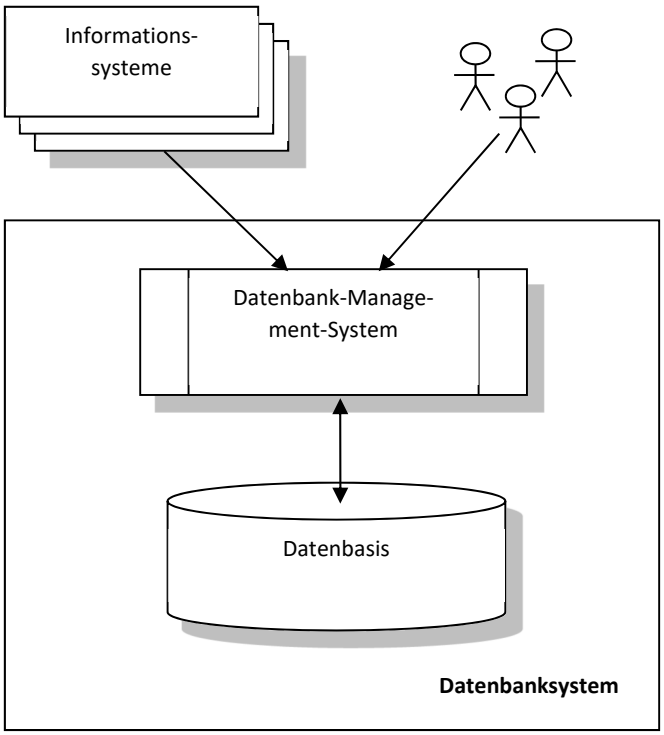
können gleichzeitig auf alle Daten eines Systems zugreifen, ohne dass es zu Beeinträchtigungen[4] kommt. Falls dieser Zugriff nicht koordiniert wird, kann es leicht zu einem Datenverlust oder einer Datenverfälschung kommen. Dateisysteme verfügen über keine oder nur sehr schwache Mechanismen, den parallelen Zugriff zu koordinieren und zu kontrollieren.

- Betriebssysteme bieten nur sehr geringe Möglichkeiten, sich gegen einen **Datenverlust** abzusichern. Dateien lassen sich zwar periodisch auf externen Datenträgern sichern. Kommt es aber zwischen zwei Sicherungen zu einem Datenverlust, so gehen unweigerlich alle Informationen verloren, die nach der letzten Datensicherung gespeichert wurden. Ein möglicher Datenverlust ist in vielen Fällen inakzeptabel und muss unter allen Umständen vermieden werden!
- Der **Datenschutz** stellt eine der wesentlichen Herausforderungen in der modernen Massendatenspeicherung dar. Eine Verletzung von Datenschutzrichtlinien kann für Unternehmen und Führungskräfte weitreichende Folgen haben. Dateisysteme erlauben nur eine sehr eingeschränkte Vergabe von Zugriffrechten.
- Das Antwortzeitverhalten eines Informationssystems hängt maßgeblich von der **effizienten Datenspeicherung** ab. Die Entwicklung von individuellen Speicherstrukturen, die dieser Anforderung genügen, ist häufig mit **hohem Kosten- und Zeitaufwand** verbunden, da sehr spezielle Hardware- und Systemkenntnisse erforderlich sind.

Es sollte offensichtlich sein, dass die Massendatenspeicherung moderner Informationssysteme auf der Basis von Dateien nicht sinnvoll realisierbar ist. Um die vorgenannten Probleme zu lösen, haben sich Datenbanksysteme als eigenständige Technologie zur Datenspeicherung und –verwaltung etabliert. Die folgende Abbildung zeigt den schematischen Aufbau eines **Datenbanksystems**[5]:

4 Eine Beeinträchtigung liegt auch dann vor, wenn ein Anwender unnötig lange Wartezeiten bei der Benutzung des Systems akzeptieren muss.

5 Die Begriffe Datenbanksysteme und Datenbank werden häufig synonym zueinander verwendet.



Ein Datenbanksystem besteht aus zwei Komponenten:
- Der **Datenbasis**, die die Gesamtheit aller in einer Datenbank gespeicherten Datenobjekte repräsentiert.
- Dem **Datenbankmanagementsystem** (**DBMS**), das eine Softwarekomponente im Datenbanksystem ist und einen kontrollierten und koordinierten Zugriff auf die Daten der Datenbasis ermöglicht. Das DBMS muss Funktionen bereitstellen, die das Definieren der Struktur von Datenobjekten, das Erzeugen, Löschen und Ändern von Datenobjekten und den lesenden Zugriff auf die Datenobjekte erlauben.

An dieser Stelle kann also die konzeptionelle Definition des Datenbankbegriffs aus dem ersten Kapitel ergänzt werden um eine informationstechnische Perspektive. Aus der Sicht der Informatik wird eine Datenbank durch ein Datenbanksystem realisiert, das wie folgt definiert werden kann:

> *Eine **Datenbank** wird durch ein Softwaresystem implementiert, das als **Datenbanksystem** bezeichnet wird. Ein Datenbanksystem setzt sich aus einem Datenbankmanagementsystem und einer Datenbasis zusammensetzt und ermöglicht die effiziente Speicherung von und den Zugriff auf große, beliebig komplex strukturierte Datenmengen.*

2.1 Datenbankbenutzer

Bei der Auseinandersetzung mit dem Thema Datenbanken macht es Sinn, sich mit den unterschiedlichen Anwender- und Nutzergruppen von Datenbanksystemen zu beschäftigen. Je nach Gruppenzugehörigkeit ist man mit unterschiedlichen Aufgaben konfrontiert und muss über besondere Fertigkeiten und spezielles Wissen verfügen:

- **Datenbankadministratoren** sind für den reibungslosen Betrieb des Datenbanksystems verantwortlich. Dazu gehört u.a.,
  - Zugriffrechte für die Datenbankbenutzer zu vergeben,
  - Datensicherungen durchzuführen,
  - Das Leistungsverhalten der Datenbank zu überwachen und zu optimieren,
  - Installation neuer Versionen und Software-Updates des Datenbanksystems durch zu führen.

  Ein Datenbankadministrator muss über sehr fundierte Systemkenntnisse im Bereich der Technischen Informatik verfügen. Damit verbunden ist das Wissen über den internen Aufbau des eingesetzten Datenbanksystems und dessen Zusammenspiel mit Systemkomponenten wie Betriebssystem und Hardware.

- **Anwendungsprogrammierer** realisieren Informationssysteme unter Einsatz von Programmiersprachen und Softwareentwicklungswerkzeugen. Da Datenbanken ein integraler Bestandteil dieser Informationssysteme sind, haben Programmierer sicherzustellen, dass
  - die Konsistenz der Datenbank durch die erstellten Programme nicht verletzt wird,
  - die in Informationssystemen abgebildeten Geschäftsprozesse durch Datenbanktransaktionen6 korrekt implementiert werden,
  - ein möglichst reibungsloser Mehrbenutzerbetrieb der Informationssysteme realisiert wird.

  Programmierer müssen daher neben fundierten Kenntnissen von Programmiersprachen und Softwarearchitekturen wissen, wie sie die Schnittstelle zwischen Programmen und Datenbanksystem implementieren.

- **Datenbankdesigner** sind für den Entwurf der Datenbank zuständig. Zu ihren wesentlichen Aufgaben gehört es
  - die Anforderungen die unterschiedlichen Endanwender zusammenzutragen,
  - eine möglichst realitätsnahe und anwenderorientierte Beschreibung der zukünftigen Datenbank zu erstellen,
  - die anwenderorientierte Sicht auf die Datenbank in eine für die Implementierung der Datenbank brauchbare Beschreibung zu transformieren.

  Datenbankdesigner müssen über sehr gute analytische und kommunikative Fähigkeiten verfügen. Sie müssen in der Lage sein, von konkreten Beispielen zu abstrahieren und zu modellieren. Zur Beschreibung von Datenbanken müssen sie verschiedene Modellierungssprachen beherrschen.

6 Unter einer Datenbanktransaktion versteht man die Ausführung einer Abfolge elementarer Datenbankaktionen, die Daten lesen, löschen, ändern oder einfügen, um einen bestimmten Geschäftsvorfall zu implementieren.

- **Endanwender** nutzen die Datenbank zur Erledigung täglicher Arbeiten im Unternehmen. Die Nutzung der Datenbank umfasst dabei
  - die Abfrage von Daten durch den Aufruf vorgegebener Reports,
  - die Auswertung von Datenbeständen nach individuellen Bedürfnissen,
  - das Aktualisieren bestehender Datenbestände.

  Die Gruppe der Endanwender lässt sich weiter differenzieren in:
  - **gelegentliche Endbenutzer**, die auf der Suche nach bestimmten Informationen auf unterschiedliche Daten zugreifen. Typischerweise gehören diese Anwender zur Management-Ebene und wünschen sich einen aggregierten Blick auf die Daten. Sie sind weniger an einzelnen Detaildaten interessiert. Zur Befriedigung ihres Informationsbedürfnisses setzen sie typischerweise analytische Informationssysteme ein. Ihre Fähigkeiten beschränken sich dabei auf die Bedienung dieser teilweise recht komplexen Informationssysteme.
  - Als **naive** oder **parametrische Endbenutzer** werden Personen bezeichnet, die die Datenbank unter Verwendung von operativen Informationssystemen benutzen. Sie setzen dieses System dazu ein, wiederkehrende Geschäftsvorfälle in der Datenbank zu verbuchen, d.h. Daten zu aktualisieren oder neue Daten hinzuzufügen[7]. Diese Klasse von Endanwender benötigt im Prinzip keine besonderen Computerkenntnisse, da sie auf ihre Aufgaben zugeschnitten Standardvorgänge mit einfach zu bedienenden Informationssystemen durchführen müssen.
  - **Professionelle Endanwender** sind in der Lage, Datenbestände nach ihren individuellen Bedürfnissen zu analysieren. Sie beherrschen eine Datenbanksprache, mit deren Hilfe sie Datenbankanfragen frei formulieren[8] können. Typischerweise gehören sie zu der Gruppe von Ingenieuren, Wirtschaftsanalysten oder Führungskräften, die für das operative Management zuständig sind. In Unternehmen trifft man professionelle Endanwender sehr häufig im Bereich von Controlling und Marketing an, aber auch in den Bereichen von Produktion und Logistik.

Die im Rahmen dieses Moduls erworbenen Kenntnisse und Kompetenzen stellen eine elementare Basisqualifikation für die Profile des Datenbankdesigners und des professionellen Endanwenders dar. Die Vermittlung systemtechnischer Kompetenz, wie sie ein Datenbankadministrator haben muss oder programmiertechnischer Kompetenz, wie sie für Anwendungsprogrammierer von Nöten ist, sind **nicht** Gegenstand dieser Lerneinheit!

7 Ein typisches Beispiel hierfür sind die Mitarbeiter im Reisebüro, die mit einem operativen Informationssystem nach Reiseangeboten suchen und Reservierung und Stornierung von Reisebuchungen durchführen.

8 Dabei sollen Kenntnisse von Programmiersprachen nicht erforderlich sein.

2.2 Aufgaben eines Datenbanksystems

Die zu Beginn des Kapitels vorgestellten Anforderungen von Informationssystemen haben belegt, dass einfache Dateien nicht zur Massendatenspeicherung geeignet sind. In Anlehnung an die von *E.F.Codd*[9] formulierten Anforderungen, muss ein Datenbanksystem

1. die **integrierte Speicherung** aller von Informationssystemen verarbeiteten Daten ermöglichen, d.h. ein Datenbanksystem muss eine Vielzahl unterschiedlicher Datenformate unterstützen. Nur so kann eine redundante Speicherung, bei der evtl. Daten auf unterschiedliche Speichermedien verteilt werden, verhindert werden.
2. sich selbst beschreiben. D.h. in der Datenbank muss ein **Katalog** (oder Inhaltsverzeichnis) vorhanden sein, aus dem hervorgeht, welche Datenobjekte gespeichert werden können und welche Informationen zu diesen Datenobjekten verfügbar sind.
3. **Operationen** zur Verfügung stellen, mit denen die Strukturen von Datenobjekte beschrieben und konkrete Datenobjekte eingefügt, geändert und gelöscht werden können. Diese Operationen müssen unabhängig von dem Datenvolumen, der Struktur der Datenobjekte oder der Komplexität der Operation effizient implementiert sein.
4. die Definition **von externen Sichten** erlauben, um die Komplexität der Gesamtdatenbank beherrschbar zu machen und ein Höchstmaß an Datenunabhängigkeit zu erzielen.
5. Funktionen zur **Zugriffskontrolle** bereitstellen, die über einen einfachen Kennwortschutz hinausgehen. Es muss möglich sein, einzelne Benutzerprofile zu definieren, über die der Zugriff auf einzelne Datenobjekte oder bestimmte Detailinformationen zu Datenobjekten eingeschränkt werden kann.
6. die **Konsistenz** des Datenbestands überwachen, um Datenbankzustände zu verhindern, die in der Anwendungswelt nicht vorkommen können.
7. den **Mehrbenutzerbetrieb** der Informationssysteme synchronisieren und verhindern, dass es zu Dateninkonsistenzen (z.B. das gegenseitige Überschreiben von Änderungen) kommt und gleichzeitig ein Höchstmaß an Nebenläufigkeit der einzelnen Benutzersitzungen gewährleistet ist.
8. es erlauben **Sequenzen von Datenbankoperationen** (**Transaktionen**) zur Realisierung von Geschäftsvorfällen zusammenzufassen. Eine solche Transaktion muss von dem Datenbanksystem als eine logische Einheit betrachtet werden und vollständig oder gar nicht ausgeführt werden. Dabei muss die Datenbank am Ende der Transaktion in einem konsistenten Zustand versetzen sein, der dauerhaft ist.
9. die **Sicherheit** der gespeicherten Daten garantieren und die Wiederherstellung eines konsistenten Datenbankzustands nach unerwarteten Fehlersituationen (z.B. Stromausfall, Systemabsturz, etc.) erlauben, ohne dass es zu einem Datenverlust kommt.

9 In einem 1982 erschienen Zeitschriftenartikel hat E.F. Codd allgemeine Anforderungen an Datenbanksysteme formuliert (siehe Literaturempfehlungen).

Die in dieser Liste dokumentierten Anforderungen werden im Prinzip von allen großen Datenbanksystemen, die im kommerziellen Umfeld eingesetzt werden, erfüllt. Datenbanksysteme, die in einem eher technischen Anwendungskontext zum Einsatz kommen, müssen nicht notwendigerweise allen der oben genannten Anforderungen genügen. Allerdings gibt aus für solche Systeme Minimalanforderrungen wie z.B. die Datensicherheit.

2.3 Phasen des Datenbankentwurfs

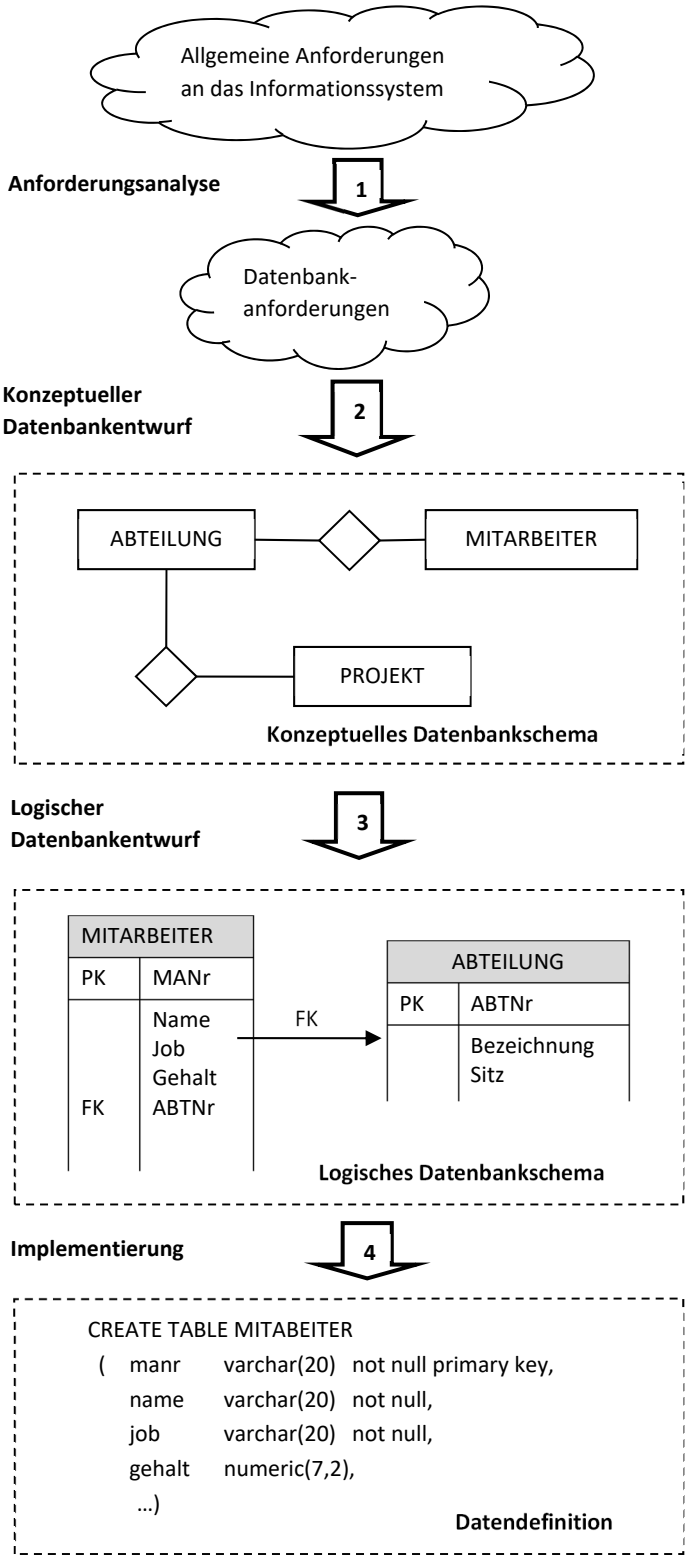
Datenbanken dienen zur Massendatenspeicherung in Informationssystemen. Daher gehören der Entwurf und die Implementierung einer sachgerechten Datenbank zu den Teilaufgaben, die im Rahmen eines Softwareentwicklungsprojektes zu erledigen sind. Betrachtet man den Prozess[10] der Softwareentwicklung, so werden typischerweise die folgenden Phasen durchlaufen:

- die **Anforderungsdefinition**, mit dem Ziel alle funktionalen und nicht-funktionalen Systemanforderungen systematisch zu erfassen und zu dokumentieren.
- die **Systemanalyse** und **-modellierung**, um aus den informell dokumentierten Anforderungen in einer (semi-)formalen Notation ein widerspruchsfreies und vollständiges Bild des zukünftigen Informationssystems zu gewinnen.
- der **System-** und **Software-Entwurf**, an deren Ende eine Systemarchitektur und eine Beschreibung der einzelnen Softwarekomponenten stehen, die mit Hilfe von konkreten Programmiersprachen, Entwicklungswerkzeugen und anderer Hilfsmittel implementiert werden können.
- die **Implementierung**, die die Umsetzung des Softwareentwurfs mit Hilfe unterschiedlicher Sprachen (z.B. Programmiersprachen) bedeutet.

Neben diesen vier Phasen gehören natürlich auch noch Testphasen, die Inbetriebnahme und die Wartung von Informationssystemen zu den Phasen im Softwareentwicklungsprozess.

Die Entwicklung einer Datenbank ist in diesen allgemeinen Prozess eingebettet und findet parallel zu den allgemeinen Phasen der Softwareentwicklung statt. In Bezug auf die Datenbank spricht man dabei von dem **Datenbankentwurf**. Das Ziel des Datenbankentwurfs ist eine Datenbank, die den Anforderungen des zu realisierenden Informationssystems in Hinblick auf die Datenverwaltung genügt. Die einzelnen Phasen des Datenbankentwurfs sind in der folgenden Abbildung dargestellt:

10 Auf den Entwicklungsprozess von Softwaresystemen wurde in der Lerneinheit „Informationssysteme“ ausführlich eingegangen.



In den einzelnen Phasen werden die Anforderungen an die Datenbank mit Hilfe unterschiedlicher Formalismen dokumentiert. Ein Dokument, das die Anforderungen an die Datenbank formal beschreibt, wird auch als **Datenbankschema** bezeichnet. Der dabei eingesetzte Formalismus heißt **Datenmodell**. D.h. ein Datenmodell dient dazu, ein Datenbankschema (also die Struktur) einer Datenbank formal zu beschreiben. In

den einzelnen Phasen des Datenbankentwurfs werden die Anforderungen an die Datenbank auf unterschiedliche Art und Weise und aus verschiedenen Perspektiven dokumentiert:

1. In der **Anforderungsanalyse** werden die Datenbankanforderungen aus der allgemeinen Anforderungsdefinition extrahiert. Die Beschreibung der **Datenbankanforderungen** erfolgt dabei informell in Form von Texten, Tabellen, Formularen, etc.
2. Auf der Basis der informell dokumentierten Datenbankanforderungen erfolgt der konzeptuelle Datenbankentwurf. Am Ende dieser Phase steht ein **konzeptuelles Datenbankschema**, das die Datenbankanforderungen formal dokumentiert. Das konzeptuelle Datenbankschema stellt die Datenbankanforderungen aus einer anwenderorientierten Sicht dar. Der konzeptuelle Datenbankentwurf ist ein kreativer Prozess, bei dem mit Hilfe einer einfachen und intuitiv verständlichen Notation Datenbankanforderungen modelliert werden. Im Kapitel 3 wird hierfür das **ER-Modell** vorgestellt.
3. Im **logischen Datenbankentwurf** wird das konzeptuelle Datenbankschema in eine Darstellung transformiert, die sich an den Strukturen eines konkreten Datenbanksystems orientiert. In dieser Phase kommt dazu mit dem **Relationen Modell** ein logisches Datenmodell zum Einsatz. Das Ergebnis des logischen Datenbankentwurfs ist ein logisches Datenbankschema, das die Datenbankstrukturen aus der Implementierungssicht beschreibt. Der logische Datenbankentwurf ist Gegenstand der Kapitel 4 und 5.
4. In der abschließenden **Datendefinition** wird mit **SQL** eine konkrete Datenbanksprache zur Implementierung des logischen Datenbankschemas verwendet. Am Ende dieser Phase sind die erforderlichen Datenstrukturen auf der Ebene eines konkreten Datenbanksystems beschrieben und können mit der Hilfe der erstellten Datendefinition direkt in der Datenbank angelegt werden. Die Datendefinition stellt dabei nur eine formale Umsetzung des logischen Datenbankschemas in eine konkrete Datenbanksprache dar.

Am Ende des Entwicklungsprozesses steht somit eine Datenbank, die in einem konkreten Datenbanksystem implementiert ist. Diese Datenbank kann dann über das Datenbank-Management-System genutzt werden. Die Nutzung kann dabei einerseits über eine interaktive Schnittstelle mit Hilfe einer Datenbanksprache erfolgen oder über Informationssysteme, in denen über das DBMS auf die Datenbasis zugegriffen wird. Der Zustand einer Datenbank lässt sich zu einem bestimmten Zeitpunkt betrachten und ergibt sich aus den zum Betrachtungszeitpunkt gespeicherten Daten. Da eine Datenbank permanenten Änderungen unterliegt, ist der **Datenbankzustand** natürlich abhängig von dem jeweiligen Beobachtungszeitpunkt.

Der Vollständigkeit halber muss an dieser Stelle erwähnt werden, dass in dem abgebildeten Entwurfsprozess der **physische Datenbankentwurf** ausgeblendet wurde. Diese Phase findet typischerweise im Anschluss an die Datendefinition statt und umfasst u.a. folgende Aufgaben:

- interne Speicherstrukturen für und Zugriffspfade auf die Daten müssen definiert werden,
- die Verteilung der Daten auf einer oder mehreren Festplatten muss festgelegt werden,
- die erwarteten Datenmengen und deren Wachstumsgrößen für einzelne Mengen von Datenobjekten müssen bestimmt werden.

Ein guter physischer Entwurf ist Voraussetzung für geringe Zugriffszeiten, eine hohe Durchsatzrate bei Änderungsoperationen und die optimale Ausnutzung des Speicherplatzes. Da der physische Datenbankentwurf zu den Aufgaben des Datenbankadministrators gehört, wird er in dem weiteren Verlauf keine Rolle spielen.

2.4 Die Datenbanksprache SQL

Das **Relationale Datenmodell** oder auch **Relationen Modell** stellt die Basis Relationaler Datenbanken dar, die einen markbeherrschenden Charakter haben. Damit ist das Relationale Datenmodell zum Quasi-Standard im Bereich der logischen Datenmodelle geworden. Die Popularität des Relationen Modells und der Erfolg Relationaler Datenbanken liegt unter anderem in der Datenbanksprache **SQL** begründet.

Die beiden folgenden Ausdrücke sind in der Datenbanksprache SQL formuliert und können so über die interaktive SQL-Schnittstelle eines relationalen Datenbanksystems eingegeben werden[11]:

- ```
  SELECT   name, job, gehalt
  FROM  MITARBEITER
  WHERE abtnr = 20 AND gehalt < 4000;
  ```

Diese `SELECT`-Anweisung ist eine Anfrage an die Datenbank, die alle Mitarbeiter bestimmt, die zur Abteilung 20 gehören und ein Gehalt von weniger als 4000 haben. Das Ergebnis der Anfrage wird dem Anwender in Tabellenform präsentiert.

Name	Job	Gehalt
Meier	Projektleiter	3200
Frohn	Programmierer	2900
Werther	Programmierer	2700

- ```
  UPDATE MITARBEITER
     SET sal = 3700
  WHERE manr = 7739;
  ```

Die `UPDATE`-Anweisung veranlasst die Datenbank das Gehalt im Datensatz des Mitarbeiters 7739 auf 3200 zu setzen.

Die Datenbanksprache SQL ist:

- an die **englische Umgangssprache** angelehnt. Dies wird durch die beiden Beispiele deutlich und belegt, wie intuitiv die Sprache ist.
- **mengenorientiert**. Ergebnisse von Datenbankanfragen sind Mengen von Datenobjekten, die genauso strukturiert sind wie die Ausgangsmengen, die die Basis der Anfrage bilden.
- **deskriptiv**. Es wird eine Ergebnismenge durch die Angabe von Eigenschaften der gesuchten Datenobjekte beschrieben.
- **mathematisches wohldefiniert**. Damit hat jede Datenbankanweisung eine eindeutige Semantik und es ist festgelegt, was die Anweisung zum Ergebnis hat. Es gibt somit keinen Interpretationsspielraum für das Datenbanksystem.
- **standardisiert**, seit 1987 wird die Sprache fortlaufend auf internationaler Ebene normiert. Aktuell liegt mit SQL:2019 die siebte Überarbeitung und der vorläufig letzte verabschiedete SQL-Standard vor.

Die Auseinandersetzung mit der Anfragesprache SQL ist Gegenstand des Kapitels 6.

11 Die Auseinandersetzung mit SQL wird in dem Kapitel 6 einen breiten Raum einnehmen. Es wird an dieser Stelle nicht erwartet, dass die abgebildeten Ausdrücke im Detail verstanden werden. Sie dienen dazu, einen ersten Eindruck der Sprache zu vermitteln und die Eigenschaften von SQL zu illustrieren.

2.5 Aktuelle Entwicklungen

Relationale Datenbanksysteme sind hervorragend geeignet, strukturierte Daten, die sich in Spalten- und Zeilenstruktur organisieren lassen, zu verwalten. Werden Datenbanken lediglich dazu verwendet, operative Geschäftstätigkeiten, wie z.B. der Eingang eines Kundeauftrags oder den Versand eines Artikels abzubilden, ist diese Form der Datenorganisation sehr gut geeignet. Die hierbei anfallenden Daten sind je nach Geschäftsvorfall gleichförmig strukturiert. Daten, die in der operativen Geschäftstätigkeit anfallen, stellen aber auch entscheidungsrelevante Informationen dar. So lassen sich aus diesen Daten z.B. Muster ableiten, die Auskunft über Kundenverhalten liefern. Aussagen, wie „Kunden die diesen Artikel gekauft haben kaufen auch …“ sind ein Beispiel für diese Art von Datenanalyse.

Der Wert der Daten und die Möglichkeit aus Daten neues Wissen abzuleiten steigt mit der Menge an Daten, die zur Verfügung stehen. Durch die fortschreitende Digitalisierung aller Lebens- und Geschäftsbereiche ist die Datenflut sprunghaft angestiegen. Neue interessante Datenquellen stellen z.B. eMail-Nachrichten, Social-Media-Inhalten und Sensoren, die in Geräten verbaut sind dar. Die Daten dieser neuen Datenquellen sind allerdings im Vergleich zu den operativen Daten unstrukturiert und passen nicht zu der strukturierten Art der Datenspeicherung in relationalen Datenbanken. Aus dieser Beobachtung heraus haben sich neue Begrifflichkeiten im Bereich des Datenmanagements herausgebildet. Hierzu gehören u.a. **Big Data** und **NoSQL**[12].

Der Begriff Big Data legt bereits nahe, dass es sich hierbei um sehr große Datenmenge handelt. Der Eigenschaften von Big Data lassen sich über das sogenannte 3-V-Modell definieren, das Big Data über die drei Dimensionen **VOLUME**, **VELOCITY** und **VARIETY** beschreibt:

12 Einen kurzen und präzisen Überblick zu dem Themenkomplex, an den sich auch die hier gemachten Ausführungen anlehnen, findet sich im Informatik-Lexikon der Gesellschaft für Informatik unter https://gi.de/informatiklexikon/big-data/.

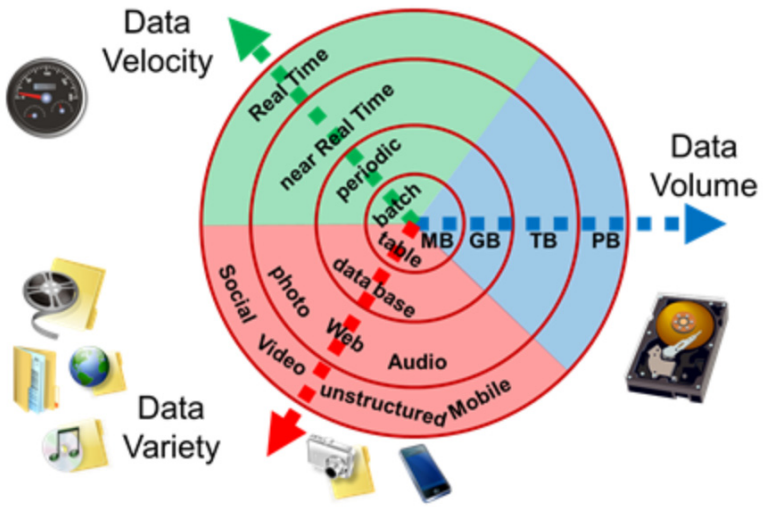
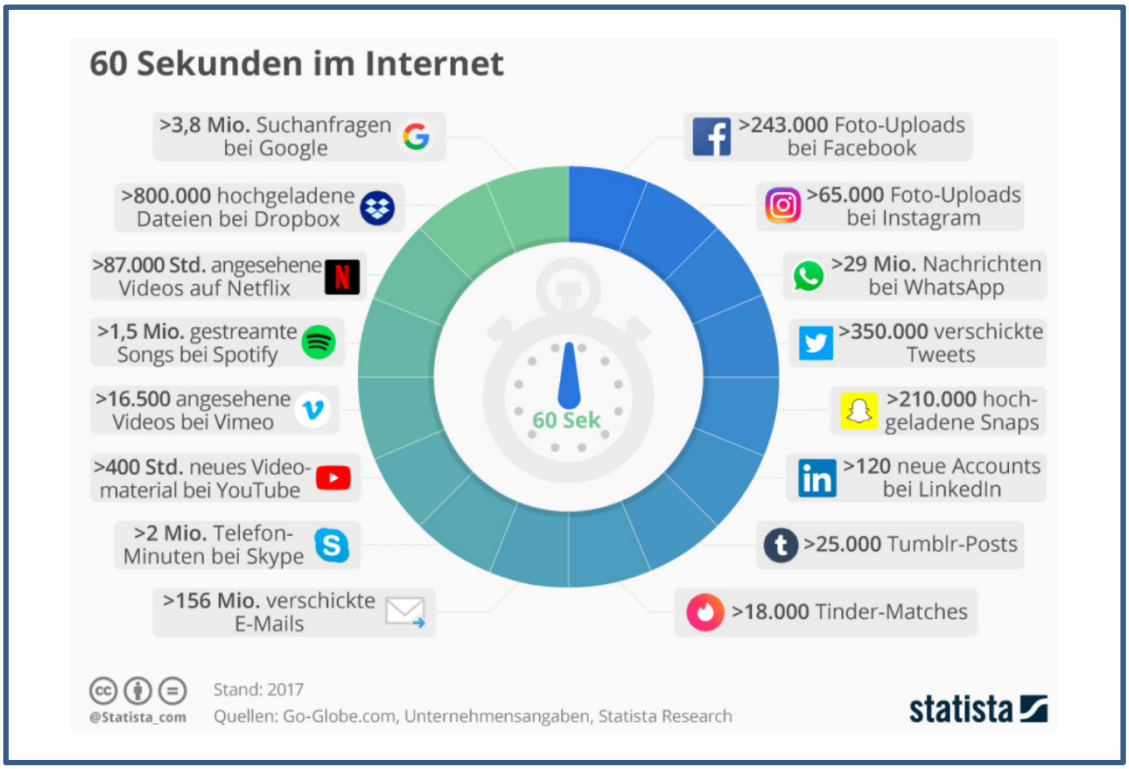


Abbildung: V-Modell für Big Data
(Quelle: https://gi.de/informatiklexikon/big-data/)

Dabei beschreibt **VOLUME** die Größe der Datenmenge. Wesentlicher Treiber ist hierbei das Internet, das zu einem exponentiellen Wachstum der Datenmenge in den letzten Jahren beigetragen hat. Die folgende Abbildung zeigt sehr anschaulich, welches Datenvolumen in 60 Sekunde im Internet kommuniziert wird.



VELOCITY umfasst neben der Geschwindigkeit, mit der Daten generiert werden, auch die Notwendigkeit, gesammelte Daten zeitnah weiterzuverarbeiten.

Das dritte **V** steht für **VARIETY** und beschreibt die Vielzahl der Datenformate, die aus den unterschiedlichen Datenquellen resultieren. Diese Daten sind nicht so klar strukturiert, wie es in transaktionalen System der Fall ist. Eine eMail kann noch als semistrukturiert beschrieben werden, da sie aus einem Nachrichtenkopf mit Sender-, Empfänger- und Betreffangabe und einem Nachrichtentext besteht. Sobald man allerdings auch mögliche Anhänge einer eMail, die wiederum jeweils ein eigenes Datenformat (PDF, DOC, JPEG, …) haben können, betrachtet, wird aus einem semistrukturierten Format ein unstrukturiertes. In dieser Eigenschaft liegt auch das wesentliche Unterscheidungsmerkmal zwischen Big Data und Business Intelligence, das in der ersten Lerneinheit vorgestellt wurde. BI geht von strukturierten Daten aus, während Big Data explizit auch semi- und unstrukturierte Daten umfasst.

Häufig finden sich in der Literatur zwei weitere **V**'s. Diese stehen für **VARICITY** und **VALUE**. Dabei steht **VARICITY** für die Zuverlässigkeit und Genauigkeit der Daten und bezieht sich somit auf die Authentizität der Datenquellen. In diesem Punkt besteht auch ein wesentlicher Unterschied zu Business Intelligence. BI-Lösungen setzen auf einem Data Warehouse auf, das konsistente und harmonisierte Daten enthält.
Mit **VALUE** kommt eine betriebswirtschaftliche Dimension in die Betrachtung, die ausdrückt, dass die Daten und die auf ihnen aufbauend Analysen einen Wert für Unternehmen schaffen müssen.

Relationale Datenbanksysteme sind in der Lage, auch große und sehr große Datenmengen zu verarbeiten. Ihr größtes Manko für die Speicherung von Big Data ist die feste Struktur, in der die Daten vorliegen müssen. Zur effizienten Speicherung von großen unstrukturierter Datenmengen haben sich neue Datenbanksysteme etabliert, die unter dem Begriff NoSQL (Not only SQL) zusammengefasst werden. NoSQL-Datenbanken eigenen sich besonders gut zur Speicherung großer Datenmengen, die keinen oder nur seltenen Änderungen unterworfen sind und die sich durch ständiges Wachstum auszeichnen. Die NoSQL-Datenbanksysteme lassen sich je nach verwendetem Datenmodell in dokumentenorientierte Datenbanken, Graphen-Datenbanken und Key-Value-Datenbanken unterteilt.

Eine Sonderstellung im Kontext von Big Data nimmt das Apache Hadoop Framework ein, das sich im Wesentlichen aus zwei Komponenten zusammensetzt:

- dem Hadoop Distributed File System (HDFS) zur verteilten Speicherung großer Datenmengen und
- dem MapReduce-Algorithmus, der eine hoch performante parallele Datenverarbeitung ermöglicht.

Big Data stellt Unternehmen vor eine Vielzahl komplexer Herausforderungen. Diese sind zunächst technischer Natur wie sie durch die 3 V's beschrieben sind und der daraus resultierende Bedarf nach einer IT-Infrastruktur, die mit den stetig wachsenden Datenmengen skalierbar ist. Aus Unternehmenssicht stellen sich neben diese technischen Anforderungen weitere komplexe Probleme, die es zu lösen gilt:

- bei der Verarbeitung von personenbezogenen Daten sind rechtliche Rahmenbedingungen zu beachten, d.h. Privatsphäre und Sicherheitsbestimmungen müssen berücksichtigt werden.
- Die Auswertung großer komplexer Datenmengen, erfordert spezielles KnowHow, das in den seltensten Fällen in IT-Abteilungen verfügbar ist.

An dieser Stelle wird der kleine Exkurs in den Bereich von Big Data und NoSQL beendet. In den folgenden Kapiteln stehen wieder die klassischen Datenbanken (Relationale Datenbanksysteme) im Mittelpunkt der Betrachtung, wie sie zur Verarbeitung operativer Daten in den meisten Unternehmen zum Einsatz kommen.

3 Konzeptueller Datenbankentwurf

In der Phase der **konzeptuelle Datenmodellierung** wird ein **konzeptueller Datenbankentwurf** erstellt, der als Diskussionsgrundlage mit dem Anwender dienen soll und gleichzeitig so formal sein muss, dass er als verbindliche Ausgangsbasis für eine nachfolgende Implementierung eingesetzt werden kann. Um diesen Anforderungen zu genügen, sollte

- von technischen Details, d.h. Implementierungskonzepten weitestgehend abstrahiert werden.
- die Darstellung einfach lesbar sein (also nicht in Form einer kryptisch erscheinenden Programmiersprache erfolgen.)
- die Informationsstrukturen so darstellbar sein, dass alle Aspekte der Anwendungswelt, die nicht die Datenbank betreffen, ausgeblendet werden können.

Aus diesen Anforderungen resultiert, dass sich zum konzeptuellen Datenbankentwurf mit dem **Entity-Relationship-Modell** (**ER-Modell**) ein Formalismus durchgesetzt hat, der eine grafische Notation verwendet und abstrakt ist, d.h. nicht auf ein bestimmtes Datenbanksystem oder eine bestimmte Technologie ausgerichtet ist.

Das Ergebnis der ER-Modellierung[13] ist ein **ER-Diagramm**, das alle möglichen Datenbankzustände festlegt, die in der modellierten Anwendungswelt sinnvoll sind. Eine spätere Implementierung der Datenbank hat somit sicherzustellen, dass nur Daten gespeichert werden können, die im Sinne des konzeptuellen Schemas zulässig sind. Jeder in der Modellierung gemachte Fehler und jede Ungenauigkeit führt zu einer Datenbank, die die Anwendungswelt nicht korrekt wiederspiegelt, d.h. in der Daten gespeichert werden können, die so nicht in der Anwendungswelt vorkommen können.

Die bisherigen Ausführungen erwecken den Eindruck, als ob es *das eine ER-Modell* gibt. Leider ist das nicht der Fall. Der ursprüngliche Vorschlag für das ER-Modell geht auf das Jahr 1976 zurück und wurde von Peter Chen[14] formuliert. In der Folge ist eine Vielzahl von Vorschlägen entstanden, die alle auf dieser Ursprungsidee aufbauen und in der Praxis eingesetzt werden

Zur ER-Modellierung existiert eine Reihe von **Softwareentwicklungs-Werkzeugen** (auch **CASE[15]-Tools** genannt), die von unterschiedlichen Unternehmen angeboten werden und in der Regel proprietäre Notationen für das ER-Modell vorschlagen. Es wird bewusst auf die Verwendung eines dieser Werkzeuge verzichtet, da nicht die Bedienung eines Tools im Vordergrund stehen soll, sondern die Konzepte des ER-Modells vermittelt werden.

---

13 Die Begriffe *ER-Modellierung, konzeptueller Datenbankentwurf* und *konzeptuelle Datenbankmodellierung* werden in dieser Lerneinheit synonym zueinander verwendet.

14 Original-Artikel im Internet unter: http://csc.lsu.edu/news/erd.pdf

15 CASE ist Abkürzung für *Computer Aided Software Engineering*

3.1 Die Fallstudie FH-Info

In Anlehnung an das **Verbundstudium an der FH Bielefeld** wird in dieser Lerneinheit eine Fallstudie verwendet, die sich mit der Organisation des Studienbetriebs in den verschiedenen Studiengängen beschäftigt. Die Fallstudie bietet eine anschauliche Arbeitsgrundlage, an der die Konzepte des ER-Modells erläutert werden, und sollte den Lesern aus ihrer Erfahrungswelt vertraut sein. Um sich nicht in unnötiger Komplexität des Beispiels zu verlieren, sind in der Fallstudie einige Sachverhalte gegenüber der Realität vereinfacht oder verändert worden.

In der Fallstudie wird davon ausgegangen, dass die FH Bielefeld eine Reihe von Verbundstudiengängen verwaltet. Hierzu ist das Informationssystem **FH-Info** zu realisieren, das von Studierenden und Dozenten als Internet-Anwendung benutzt werden soll. FH-Info soll später u.a. Auskunft über folgende Sachverhalte geben:

- Welche Studiengänge werden angeboten und wie sind diese Studiengänge in Form von Modulen aufgebaut?
- Welche Lehrveranstaltungen werden von welchen Dozenten angeboten?
- Welche Studierenden und welche Dozenten sind in einem Studiengang aktiv?
- Welche Lerneinheiten gehören zu welchen Modulen und wer hat diese Lerneinheiten verfasst?

Damit diese Fragen durch das System beantwortet werden können, müssen natürlich die notwendigen Informationen in der zugehörigen Datenbank gespeichert werden. Im Verlauf dieser Lerneinheit werden daher ein konzeptuelles Datenbankschema für diese Fallstudie erstellt und dabei die Konzepte des ER-Modells erläutert.

Zum Verständnis der folgenden Teilkapitel ist es erforderlich die im Anhang befindliche FH-Info-Fallstudie zu lesen!

3.2 Basiskonzepte des ER-Modells

Bei der Vorstellung der elementaren Konzepte des ER-Modells lehnt sich dieses Kapitel im Wesentlichen an die ursprünglich im Jahr 1976 von Peter Chen[16] vorgeschlagene Notation an.

Das Ergebnis der ER-Modellierung ist ein **ER-Diagramm**, das alle Sachverhalte der Anwendungswelt beschreibt, die später in der Datenbank gespeichert werden sollen. Auch wenn das ER-Modell es nicht explizit verlangt, macht es Sinn, jedem ER-Diagramm einen Titel zu geben, so dass sich dem Betrachter erschließt, welche Anwendungswelt dargestellt ist. In der verwendeten Fallstudie wäre „*FH-Info-Datenbank*“ ein möglicher Diagrammtitel.

16 Der Originalartikel ist unter http://csc.lsu.edu/news/erd.pdf zu finden.

Zwei der drei grundlegenden Konzepte des ER-Modells bilden dessen Namen:

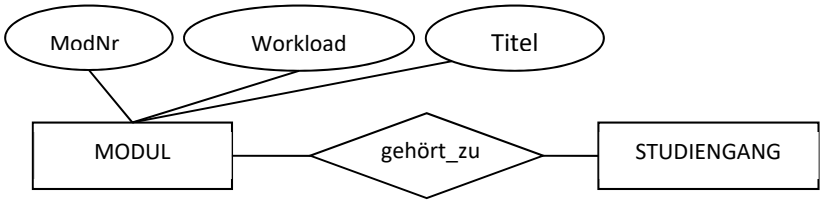
- Ein **Entity** steht für ein reales oder fiktives Objekt der Anwendungswelt, über das etwas gespeichert werden soll.
- Der Begriff **Relationship** bezeichnet eine konkrete Beziehung, die zwischen Objekten der Anwendungswelt besteht.

Als drittes Konzept kennt das ER-Modell noch

- Das **Attribut**, das für eine Eigenschaft eines Objektes oder einer Beziehung steht, die gespeichert werden soll.

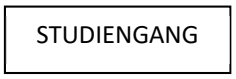
Ein Beispiel für ein *Entity* in der FH-Info-Fallstudie sind ein konkreter Studiengang (z.B. *Betriebswirtschaftslehre*) oder ein Veranstaltungsmodul (z.B. *Accounting*). Für eine *Beziehung (Relationship)* ist die Zuordnung des Moduls *Accounting* zum Studiengang *Betriebswirtschaftslehre* ein Beispiel. Attribute des Moduls (*Accounting*) sind dessen Modulnummer (*4711*) und dessen Workkload (*5 Credit Points*).

Jedes dieser drei Konzepte lässt sich im ER-Modell grafisch darstellen:

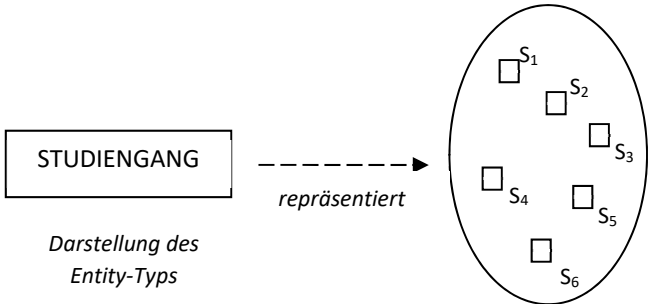


3.2.1 Entity-Typen

Ein **Entity-Typ** repräsentiert eine Menge von Objekten, die über die gleichen charakteristischen Eigenschaften beschrieben werden können (z.B. hat jeder Studiengang eine Bezeichnung und einen Abschluss). In einem ER-Diagramm wird ein Entity-Typ durch ein Rechteck dargestellt, in das der Name der Objektmenge eingetragen wird:



Bei der Modellierung wird nicht jedes konkrete Objekt der Anwendungswelt einzeln dargestellt, sondern es werden Schablonen erstellt, die eine Menge von gleichförmigen Objekten beschreiben[17].



Darstellung des Entity-Typs

Objektmenge mit aktuell 6 Studiengangsobjekten

Jedes ER-Diagramm setzt sich aus einer Vielzahl von Entity-Typen zusammen[18].

3.2.2 Werte und Attribute

Zu jedem Objekt der Anwendungswelt sind bestimmte Informationen zu speichern. Diese Informationen werden in Form von **Werten** in der Datenbank abgelegt (z.B. `1104711` für die konkrete Matrikelnummer eines Studierenden). Zur Darstellung von Werten werden im ER-Modell Wertebereiche verwendet, die die Mengen möglicher Werte beschreiben:

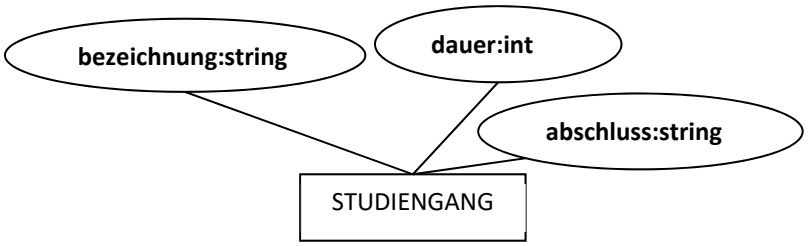
17 Die dargestellte Objektmenge stellt einen möglichen Datenbankzustand dar, der auch als eine mögliche Ausprägung des Entity-Typs bezeichnet wird.

18 Die oben abgebildete Ausprägung (Objektmenge) ist natürlich nicht Bestandteil des ER-Diagramms.

Tabelle 1: Bezeichnung und Interpretation von Wertebereichen

Bezeichnung	Werte
`string`	für Texte beliebiger Länge
`date`	zur Darstellung von Datumswerten
`time`	für Zeitwerte
`int`	für ganze Zahlen
`float`	für rationale Zahlen

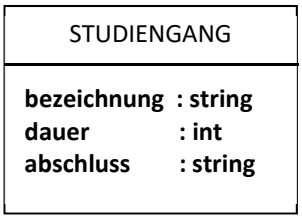
Jeder Entity-Typ verfügt über **Attribute**, die die Eigenschaften beschreiben, die zu einem Entity gespeichert werden sollen. In der klassischen ER-Notation werden Attribute durch Ovale dargestellt, die mit einem Attributbezeichner und der Angabe eines Wertebereichs beschriftet und durch eine Linie mit dem zugehörigen Entity-Typ verbunden sind:



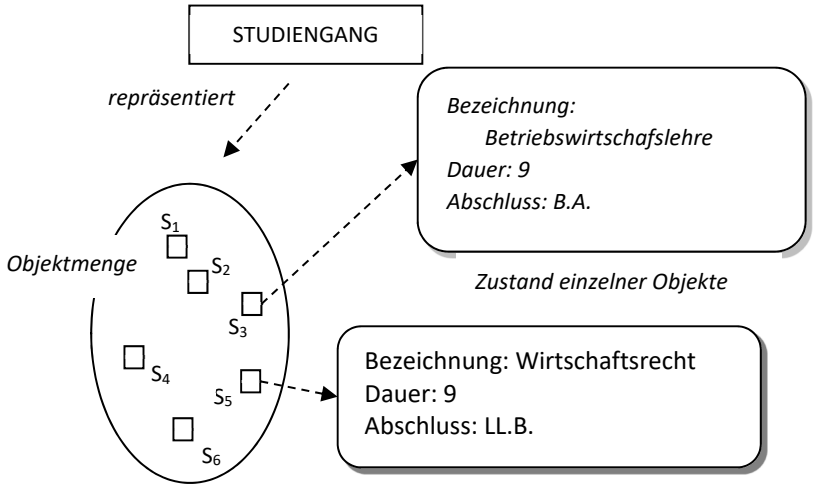
Das ER-Modell schreibt keinen festen Satz an Wertebereichen vor. Je nach Anwendungsgebiet sind beliebige Wertebereiche denkbar. In der Praxis ist es allgemein üblich, auf die Angabe von Wertebereichen verzichtet.

Der Zustand eines Datenbankobjektes (d.h. eines Entites) wird durch die aktuellen Werte seiner Attribute bestimmt. Dabei kann jedes Attribut eines Entities nur genau einen Wert aufweisen.

Als Alternative zu der Notation der Attribute in einem Oval hat sich die folgende Darstellung etabliert:

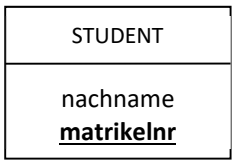


Hierbei werden die Attribute in einem Rechteck unterhalb der Bezeichnung des Entity-Typs notiert. Da diese Notation etwas kompakter ist, wird sie im weiteren Verlauf Verwendung finden.



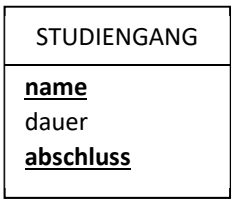
Schlüssel und Schlüsselattribute

In einer Datenbank lassen sich nicht die konkreten Objekte der Anwendungswelt speichern. Es werden vielmehr Informationen (d.h. Werte) über die Objekte dargestellt, bzw. abgelegt. Damit ergibt sich das grundsätzliche Problem der eindeutigen Zuordnung von Informationen aus der Datenbank zu den konkreten Objekten der Anwendungswelt. Würde für einen Studierenden nur der Name gespeichert, könnte es ein Datenbankobjekt geben, dem sich zwei reale (Studenten-) Objekte zuordnen lassen. Wird als zusätzliches Attribut die *Matrikelnummer* aufgenommen, ist die eindeutige Identifizierung eines Studenten möglich. In diesem Fall ist das Attribut *Matrikelnummer* Schlüssel des Entity-Typs und wird durch Unterstreichen gekennzeichnet:



Der Schlüssel eines Entity-Typs kann auch aus mehreren Attributen bestehen. Im Fall des Entity-Typs Studiengang ist die Bezeichnung alleine nicht Schlüssel, da es zwei Studiengänge mit der Bezeichnung *Wirtschaftsrecht* geben kann. Diese beiden unterscheiden sich allerdings im Abschluss (LL.B. und LL.M.). Es kann keine zwei Studiengänge geben, die sowohl in Bezeichnung und Abschluss übereinstimmen. D.h. durch die Kombination von Bezeichnung und Abschluss ist die eindeutige Identifikation wieder gegeben. Somit bildet die Attributkombination den Schlüssel[19]:

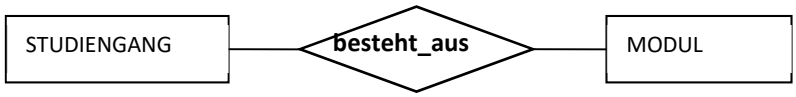
19 Auch in diesem Beispiel hätte man einen künstlichen Schlüssel *studiengangsnummer* einführen können.



3.2.3 Beziehungstypen

Analog zu der Beschreibung von Entites durch Entity-Typen werden Beziehungen zu **Beziehungstypen** modelliert. Es wird also nicht die konkrete Beziehung zwischen einem Modul (z.B. *Accounting*) und einem Studiengang (z.B. *Betriebswirtschaftslehre*) dargestellt. Stattdessen wird nur die Tatsache beschrieben, dass die Objekte eines Entity-Typs mit Objekten eines zweiten Entity-Typs in Beziehung stehen können.

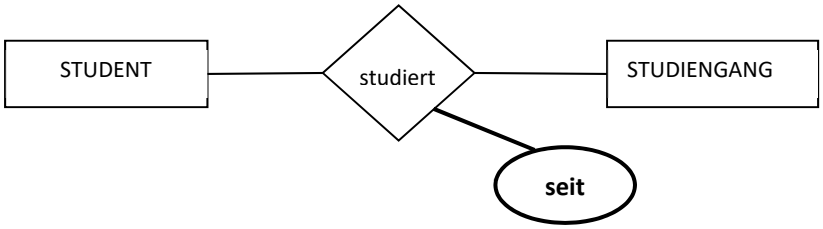
In der ER-Notation werden Beziehungstypen durch Rauten dargestellt, die mit einem Bezeichner beschriftet sind, die ein intuitives Verständnis ermöglichen. Die Raute ist über Linien mit den beteiligten Entity-Typen verbunden[20]:



Das dargestellte Beispiel drückt aus, dass Module zu Studiengängen gehören und das Studiengänge aus Modulen bestehen, d.h. eine Beziehung ist grundsätzlich bidirektional zu verstehen[21].

Beziehungsattribute

Beziehungen können über Eigenschaften verfügen, die dann durch Attribute dargestellt werden:



Für jede und jeden Studierende und Studierenden muss gespeichert werden, seit wann sie oder er einen bestimmten Studiengang studiert. Da ein Student mehrere Studiengänge studieren kann, macht es keinen

20 In dieser und allen folgenden Abbildungen wird auf die Darstellung von Attributen verzichtet, falls sie für das Verständnis des dargestellten Beispiels nicht relevant sind.

21 Der Studiengang S1 besteht also aus den Module M1 und M3 bzw. das Modul M3 gehört zu den Studiengängen S1 und S2.

Sinn, diese Information als Attribut von Student zu speichern. Die Information, seit wann ein Student einen Studiengang studiert, kann nur sinnvoll mit der Beziehung zusammen gespeichert werden.

Beziehungstypen verfügen nie über Schlüsselattribute, da eine Beziehung eindeutig über die beteiligten Entity-Typen bestimmt wird.

3.3 Kardinalitätiten

Durch die bisher eingeführte Notation für Beziehungstypen wird die Anzahl der tatsächlichen Beteiligungen eines Objektes an einer Beziehung nicht eingeschränkt:

Die Beziehung `besteht_aus` erlaubt, dass jeder Studiengang aus mehreren Modulen bestehen kann und dass jedes Modul zu mehreren Studiengängen gehören kann. Dies entspricht aber nicht der Realität, da jedes Modul nur zu einem Studiengang gehört. Somit erlaubt die obige Modellierung unzulässige Datenbankzustände.

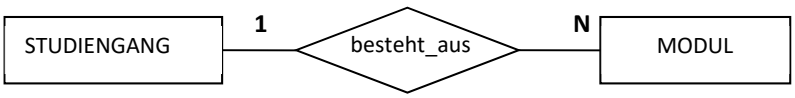
Um Beziehungstypen weiter einschränken zu können, müssen sie um **Kardinalitätsangaben** erweitert werden. In der Literatur ist eine Vielzahl von Notationen für Kardinalitätsangaben eingeführt worden. Es wird an dieser Stelle nur die Chen-Notationen vorgestellt.

1:N-Beziehungen

Bei einer 1:N-Beziehng dürfen jedem Entity des ersten Entity-Typs mehrere Entitys des zweiten Typs zugeordnet werden und jedem Entity des zweiten Typs höchstens ein Entity des ersten. Die Beziehung `besteht-aus` ist ein Beispiel für eine **1:N-Beziehung**, da

> *Jeder Studiengang aus mehreren Modulen bestehen kann und jedes Modul zu nur einem Studiengang gehören darf.*

Diese Obergrenzen werden in der Chen-Notation wie folgt ausgedrückt:

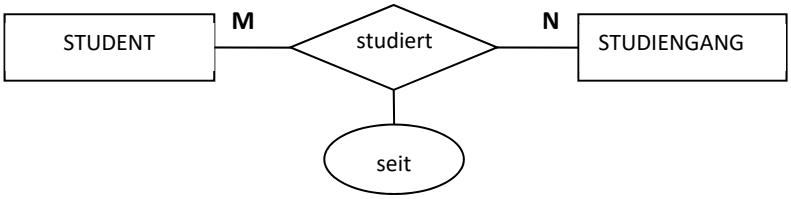


Die verwendete Notation berücksichtigt die Leserichtung:
- Jeder Studiengang kann aus (`N`) mehreren Modulen bestehen.
- Jedes Modul gehört zu höchstens (`1`) einem Studiengang.

N:M-Beziehungen

Bei einer N:M-Beziehung werden die Obergrenzen nicht eingeschränkt. D.h. jedes Entity des ersten Typs kann mit beliebig vielen des zweiten Typs in Beziehung stehen und umgekehrt. Die Beziehung *bietet-an* ist ein Beispiel für eine N:M-Beziehung, da

- *Jeder Student kann mehrere (N) Studiengänge studieren und*
- *Jeder Studiengang von mehreren(M) Studierenden studiert werden kann.*



Die Symbole **N** und **M** stehen hierbei für eine beliebige Anzahl.

1:1-Beziehungen

Bei einer 1:1-Beziehung werden die Obergrenzen in beide Leserichtungen auf Eins gesetzt. Die Beziehung *ist-für* zwischen *Akkreditierung* und *Studiengang* ist ein Beispiel für eine 1:1-Beziehung, da

- *Jeder Studiengang nur eine* (1) *Akkreditierung haben kann und*
- *Jede Akkreditierung für nur einen* (1) *Studiengang ist.*

3.4 Aufgaben

3.4.1 Fallstudie FH-Info

Erstellen Sie das vollständige ER-Diagramm zu der im Anhang vorgestellten Fallstudie FH-Info.

Für diese Fallstudie wird bewusst auf die Angabe einer Musterlösung in dieser Lerneinheit verzichtet. Bitte bringen Sie ihren Lösungsvorschlag mit in die Präsenzveranstaltung. Dort wird Zeit und Raum sein, sich über unterschiedliche Modellierungsalternativen zu diskutieren und zu bewerten.

3.4.2 Fallstudie eLibri

Eine Reihe unabhängiger Buchhändler hat sich zu der Einkaufgenossenschaft *eLibri* zusammengeschlossen. *eLibri* ist die Antwort dieser Buchhändler auf die erdrückende Konkurrenz aus dem Internet durch Anbieter wie z.B. *amazon*.

eLibri betreibt ein großes Logistikzentrum im Großraum Frankfurt, über das alle angeschlossenen Buchhändler mit Büchern versorgt werden. Außerdem tritt *eLibri* mit einer E-Commerce-Plattform im Internet auf. Über diese Plattform können registrierte Privatkunden Bücher bestellen. Als einzige Zahlungsmethode bietet *eLibri* den Privatkunden die Zahlung per Kreditkarte an.

Die Projektaufgabe besteht darin, einen konzeptuellen Entwurf der Datenbank des *eLibri*-Systems mit Hilfe des ER-Modells zu erstellen. Dieses System soll die folgenden Sachverhalte abbilden:

Im Mittelpunkt stehen die Kundenaufträge. Jeder Kundenauftrag erhält eine eindeutige Auftragsnummer und setzt sich aus einer Reihe von Auftragspositionen zusammen. Für jeden Kundenauftrag wird das Eingangsdatum im System verwaltet. Jede Position erhält eine eindeutige Positionsnummer und ist einem Auftrag zugeordnet. Jeder Kundenauftrag verfügt über eine oder mehrere Auftragspositionen. Zu jeder Position ist das bestellte Buch im System ersichtlich. Da in einer Position von einem Buch auch mehrere Exemplare bestellt sein können, muss die Bestellmenge zu jeder Position gespeichert werden. In seltenen Fällen kommt es vor, dass eine Auftragsposition nicht ausgeliefert werden kann, da der bestellte Artikel nicht verfügbar ist. Das System muss für jede Position Auskunft über den Lieferstatus (*in Bearbeitung*, *nicht lieferbar*, *ausgeliefert*) geben können. Nicht lieferbare Positionen werden nicht nachgeliefert.

Zu allen von *eLibri* angebotenen Büchern verwaltet das System eine eindeutige Artikelnummer, eine Bezeichnung, einen Preis und den aktuellen Lagerbestand.

Die Kunden von *eLibri* teilen sich in zwei unterschiedliche Gruppen. Es gibt die Buchhändler, die an das eLibri-System angeschlossen sind und die Privatkunden, die über die E-Commerce-Plattform bestellen können.

Jeder Privatkunde hat eine eindeutige Kundennummer und eine Anschrift (bestehend aus Postleitzahl, Ort, Straße und Hausnummer). Für Privatkunden verwaltet das System außerdem den Vornamen, Namen und die Kreditkarteninformation (eindeutige Nummer, Kredikartenunternehmen und Ablaufdatum der Gültigkeit), über die die Bezahlung erfolgt. Jeder Privatkunde muss über mindestens eine Kreditkarte verfügen. Es kommt vor, dass eine Kreditkarte von mehreren Kunden (z.B. Ehepartnern) gemeinsam genutzt wird. Einige Kunden haben mehrere Kreditkarten registriert.

Für jeden Händler gibt es eine spezielle Händlernummer. Händler erhalten grundsätzlich eine Rechnung, die sie bis zum Erreichen des vereinbarten Zahlungsziels per Überweisung zu begleichen haben. Für Händler werden daher der Name des Unternehmens und das Zahlungsziel in Tagen gespeichert.

Jeder Kunde (Privatkunde oder Händler) kann mehrere Aufträge bei eLibri haben. Jeder Kundenauftrag ist genau einem Kunden zugeordnet.

Im weiteren Verlauf der Lerneinheit wird diese Fallstudie in einer weiteren Aufgabe wieder aufgegriffen. Sie finden eine mögliche Lösung für diese Modellierung am Ende des Kapitels 5.

4 Das Relationen Modell

Das in dem vorhergehenden Kapitel vorgestellte ER-Modell zeichnet sich dadurch aus, dass es intuitiv verständlich ist und gleichzeitig über eine hohe Ausdrucksstärke verfügt. Leider fehlt dem ER-Modell eine Sprachkomponente, über die Anfragen oder Änderungsoperationen formuliert werden können. Beides wird aber bei einer konkreten Implementierung einer Datenbank benötigt. Aus diesem Grund hat sich zur Implementierung von Datenbanken das **Relationen Modell** als Standard etabliert. Das Relationen Modell nimmt eine marktbeherrschende Stellung im Bereich kommerzieller Datenbanksysteme ein. Mit der Datenbanksprache **SQL** existiert zudem ein Sprachstandard für Relationale Datenbanksysteme

4.1 Grundlagen

Namensgeber des Datenmodells ist die Relation. In der Mathematik ist die Relation ein wohldefinierter Begriff, der auf der Mengentheorie aufbaut. Anschaulich kann eine **Relation** durch eine Tabelle dargestellt werden:

MITARBEITER				
MANr	**Name**	**Tätigkeit**	**Gehalt**	**Provision**
7739	Meier	Projektleiter	3200	**NULL**
7900	Karl	Manager	7200	**NULL**
8801	Langer	Vertrieb	2900	**1200**
7656	Hansen	Geschäftsführer	9800	**NULL**
7329	Frohn	Programmierer	2900	**NULL**
7722	Werther	Programmierer	2200	**NULL**
7643	Kurt	Vertrieb	3100	**900**
7767	John	Administrator	4900	**NULL**
7800	Beier	Trainer	2700	**NULL**
7621	Klaas	Trainer	3200	**NULL**

In dieser Relation werden Mitarbeiterdaten von *BestTec*[22], einem weltweit führenden Unternehmen im Bereich der Automation Technology, dargestellt. Jeder Mitarbeiter hat eine Nummer, einen Namen, übt eine Tätigkeit aus und bezieht ein Gehalt.

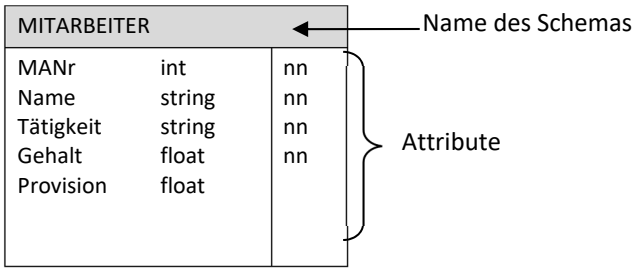
Jede Zeile in der Tabelle stellt einen **Datensatz** dar, der ein Objekt (also einen Mitarbeiter) der Anwendungswelt repräsentiert. Die Spalten in der Tabelle werden als **Attribute** der Relation bezeichnet. Jeder Datensatz weist für jedes Attribut einen konkreten Wert auf.

22 Die Fallstudie *BestTec* wird im Anhang vorgestellt und wird im Mittelpunkt der folgenden Kapitel stehen. An dieser Stelle ist es noch nicht erforderlich, die Beschreibung der Fallstudie genauer zu lesen.

Das Relationenmodell kennt **optionale Attribute**. Ein optionales Attribut kann in einem Datensatz undefiniert sein, d.h. es hat keinen konkreten Wert des ausgewiesenen Wertebereichs. Im Relationenmodell wird dieser undefinierte Zustand durch den **NULL**-Wert charakterisiert. In der Darstellung einer Relation wird für den undefinierten Wert entweder das Wort **NULL** geschrieben oder es wird kein Wert an der Stelle des Attributs angegeben:

> Das Attribut Provision gibt an, welchen variablen Gehaltsanteil ein Mitarbeiter bezieht. Für einige Mitarbeiter ist ein variabler Gehaltsanteil nicht vorgesehen. In diesen Fällen steht dort der **NULL**-Wert.

Jede Relation wird über ein **Relationenschema** beschrieben, das grafisch wie folgt notiert werden kann:



Ein Relationenschema besteht aus einem Namen und einer Liste von Attributen. Jedes Attribut wird durch einen Namen und die Angabe eines Wertebereichs definiert. Im Relationenmodell sind nur atomare Wertebereiche zulässig. Beispiele für solche atomaren Wertebereiche sind Zeichenketten, Zahlen und Datumswerte. Unzulässig sind z.B. Wertebereiche, die für Mengen oder Listen stehen.

Durch den Zusatz **nn** (**not null**) werden diejenigen Attribute markiert, die nicht optional sind, d.h. die immer einen gültigen Wert aufweisen müssen. Alle nicht markierten Attribute sind somit optional.

Eine **Relationale Datenbank** besteht aus einer Menge von Relationen. Jede Relation stellt eine konkrete Objektmenge dar und wird durch ein Relationenschema definiert.

4.2 Schlüssel

Analog zum ER-Modell gibt es im Relationen Modell das Konzept des Schlüssels. Da eine Relation eventuell mehrere alternative Schlüssel besitzen kann, werden alle möglichen Schlüssel einer Relation als **Schlüsselkandidaten** bezeichnet. Ein Schlüsselkandidat (oder kurz **Schlüssel**) ist eine minimale Menge von Attributen, die jeden Datensatz der Relation eindeutig identifiziert. D.h. es kann keine zwei Datensätze in einer Relation geben, die in den Werten dieser Attribute übereinstimmen.

Aus der Menge der Schlüsselkandidaten muss für die Implementierung einer als **Primärschlüssel** ausgewählt werden. Ein Primärschlüssel darf dabei nur nicht optionale Attribute enthalten. Bei der Notation eines Relationenschemas kennzeichnet man den Primärschlüssel durch die Angabe von **PK** (**Primary Key**). Alle weiteren Schlüsselkandidaten werden als **UK** (**Unique Key**) gekennzeichnet und gegebenenfalls durchnummeriert. Der Primärschlüssel wird abgesetzt von den restlichen Attributen notiert.

Jeder Mitarbeiter hat eine eindeutige Mitarbeiternummer und eine E-Mail-Adresse. Somit gibt es für die Mitarbeiter-Relation zwei Schlüsselkandidaten:

MITARBEITER			
PK	MANr	integer	nn
	Name	string	nn
	Tätigkeit	string	nn
	Gehalt	float	nn
	Provision	float	
UK	Email	string	nn

Im Gegensatz zu einem Primärschlüssel darf ein als Unique Key gekennzeichneter Schlüssel optionale Attribute enthalten.

Für jede Relation muss ein Primärschlüssel definiert werden. Ein relationales Datenbanksystem überwacht die Einhaltung der Schlüsseleigenschaft. D.h. es ist nicht möglich, zwei Datensätze in einer Relation zu speichern, die in den Werten der Attribute eines Schlüssels übereinstimmen. Diese Eigenschaft einer Relation wird als **Entity**- oder **Schlüsselintegrität** bezeichnet.

Attribute, die Teil eines Schlüsselkandidaten sind, werden als **Schlüsselattribute** bezeichnet. Alle anderen Attribute sind **Nichtschlüsselattribute**.

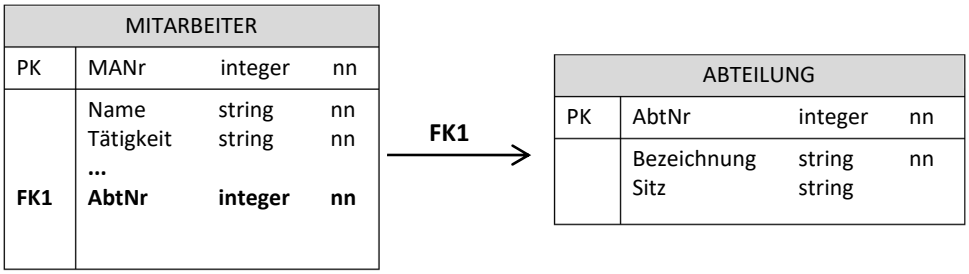
4.3 Fremdschlüssel

Beziehungen zwischen Datensätze müssen über Attribute dargestellt werden. Damit ein Datensatz auf einen anderen Datensatz verweisen kann, werden dessen Schlüsselattribute als zusätzliche Attribute aufgenommen.

Da jeder Mitarbeiter einer Abteilung zugeordnet werden muss, wird der Primärschlüssel der Relation ABTEILUNG als nicht optionales Attribut in das Relationenschema MITARBEITER aufgenommen:

MITARBEITER				
MANr	**Name**	**Tätigkeit**	...	**AbtNr**
7739	Meier	Projektleiter	...	**20**
7900	Karl	Manager	...	**20**
8801	Langer	Vertrieb	...	**10**
7656	Hansen	Geschäftsführer	...	**10**
7329	Frohn	Programmierer	...	**20**
7722	Werther	Programmierer	...	**20**
7643	Kurt	Vertrieb	...	**20**
7767	John	Administrator	...	**30**
7800	Beier	Trainer	...	**50**
7621	Klaas	Trainer	...	**50**

ABTEILUNG		
AbtNr	**Bezeichnung**	**Sitz**
10	Zentrale	Gütersloh
20	Engineering	Verl
30	Automotive	Paderborn
40	Robotik	Bielefeld
50	Education	Münster

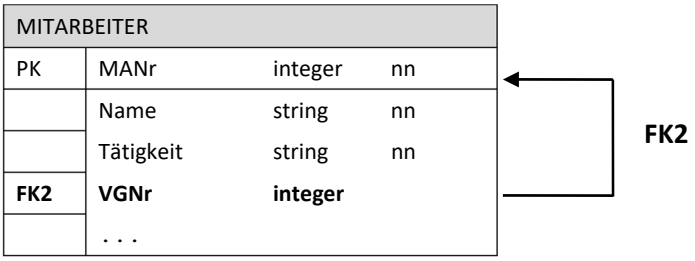


Das in die Relation MITARBEITER aufgenommene Attribut AbtNr ist der Schlüssel einer anderen (fremden) Relation und wird daher als **Fremdschlüssel** (**Foreign Key**) bezeichnet und mit dem Zusatz **FK** notiert. Da es mehrere Fremdschlüssel in einer Relation geben kann, wird dieser Zusatz gegebenenfalls mit einer laufenden Nummer versehen. Durch das Hinzufügen eines Pfeils wird die Fremdschlüsselbeziehung visualisiert.

Der Pfeil zeigt dabei immer in die Richtung der Relation, in der sich alle weiteren Informationen zu dem Datenobjekt finden lassen, das durch das Fremdschlüsselattribut referenziert wird. D.h. der Pfeil für FK1 zeigt von der MITARBEITER-Relation zur ABTEILUNG-Relation, da sich mit dem Wert des Fremdschlüsselattributes AbtNr aus der MITARBEITER-Relation in der referenzierten ABTEILUNG-Relation alle weiteren Abteilungsinformationen auffinden lassen.

Ein Fremdschlüssel kann auch Beziehungen zwischen Datensätzen einer Relation ausdrücken. Dazu wird die Mitarbeiterrelation um ein Attribut VGNr erweitert. Dieses Attribut beinhaltet die Mitarbeiternummer des Vorgesetzten und ist somit ein Fremdschlüssel, der auf den Vorgesetzten-Mitarbeiter verweist:

MITARBEITER				
MANr	**Name**	**Tätigkeit**	**VGNr**	**...**
7739	Meier	Projektleiter	**7900**	...
7900	Karl	Manager	**7656**	...
8801	Langer	Vertrieb	**7656**	...
7656	Hansen	Geschäftsführer	**NULL**	...
7329	Frohn	Programmierer	**7900**	...
7722	Werther	Programmierer	**7900**	...
7643	Kurt	Vertrieb	**7656**	...
7767	John	Administrator	**7656**	...
7800	Beier	Trainer	**7900**	...
7621	Klaas	Trainer	**7900**	...



Im Zusammenhang mit Fremdschlüsseln sollten folgende Punkte beachtet werden:

- Ein Fremdschlüssel kann sich aus mehreren Attributen zusammensetzen.
- Ein Fremdschlüssel verweist grundsätzlich auf den Primärschlüssel der referenzierten Relation!
- Ein Fremdschlüssel und der referenzierte Schlüssel müssen identische Wertebereiche aufweisen.
- Die Bezeichnungen der Fremdschlüsselattribute müssen nicht identisch mit denen der referenzierten Schlüsselattribute sein. Aus praktischen Gründen empfiehlt sich jedoch eine Namensgleichheit.
- Ein Fremdschlüssel kann eine Referenz auf einen Datensatz der gleichen Relation darstellen. In diesem Fall können Schlüssel- und Fremdschlüsselattribute nicht namensgleich sein.

Ein Fremdschlüsselattribut ist ein „normales Attribut“. D.h. es kann als „nicht optional“ (nn) definiert werden. Genauso kann er auch Teil des Schlüssels einer Relation sein.

Ein relationales Datenbanksystem überwacht die Einhaltung der Fremdschlüsseleigenschaft. D.h. zu jedem Fremdschlüsselattributwert (z.B. AbtNr in MITARBEITER) muss es auch einen entsprechenden Datensatz in der referenzierten Relation mit entsprechendem Schlüsselwert (also AbtNr in ABTEILUNG) geben. Diese Eigenschaft einer Relation wird als **Fremdschlüsselintegrität** bezeichnet.

5 Logischer Datenbankentwurf

Im Datenbankentwurf wird zunächst ein konzeptuelles Datenbankschema mit den Konzepten des ER-Modells erstellt. Ein solches Datenbankschema kann jedoch nicht direkt implementiert werden. Zunächst muss der konzeptuelle Datenbankentwurf in ein relationales Datenbankschema transformiert werden. Das so generierte relationale Schema bezeichnet man auch als **logisches Datenbankschema**.

Das Ziel der Schematransformation ist es, möglichst alle Sachverhalte, die im konzeptuellen Schema dargestellt sind, in das logische Schema zu übertragen. Bei einer exakten Transformation sollten im logischen Datenbankschema genau die gleichen Datenbankzustände (also nicht mehr und nicht weniger) wie im konzeptuellen Schema möglich sein. Falls dies gelingt, spricht man von der **Erhaltung der Informationskapazität**.

Die Schema-Transformation ist kein kreativer Prozess. Es sollen vielmehr Regeln angewendet werden, nach denen diese Abbildung rein schematisch erfolgt. D.h. ein Rückgriff auf die Anforderungsdefinition, die der Ausgangspunkt für die konzeptuelle Modellierung war, findet nicht statt.

In diesem Kapitel werden für die unterschiedlichen Konzepte des ER-Modells Transformationsregeln aufgestellt. Zu jeder Regel wird basierend auf der FH-Info-Fallstudie ein Beispiel angegeben.

5.1 Transformation von Entity-Typen

Im ersten Schritt werden alle Entity-Typen nach folgender Regel transformiert:

Regel 1: Abbildung einfacher Entity-Typen

- Jeder Entity-Typ wird auf ein Relationenschema abgebildet[23]. Dabei ergeben sich die Attribute des Relationenschemas aus denen des Entity-Typs. Für Attribute, für die im ER-Diagramm kein Wertebereich definiert wurde, wird jetzt ein Wertebereich festgelegt.
- Für alle nicht optionalen Entity-Attribute wird der `NOT-NULL`-Vermerk hinzugefügt.
- Der Primärschlüssel ergibt sich aus dem Schlüssel des Entity-Typs. Falls der Schlüssel des Entity-Typs aus mehreren Attributen zusammengesetzt ist, wird ein künstlicher Schlüssel[24] eingefügt und zum Primärschlüssel erklärt. Dieses künstliche Schlüsselattribut sollte numerisch sein. Der im ER-Diagramm definierte Schlüssel wird dann als zusätzlicher Schlüssel (`UK`) definiert.

23 Falls der Name des Entity-Typs sehr lang ist, empfiehlt es sich, eine abgekürzte Bezeichnung zu verwenden. Für lange Attributbezeichner sollten ebenfalls sinnvolle Abkürzungen verwendet werden.

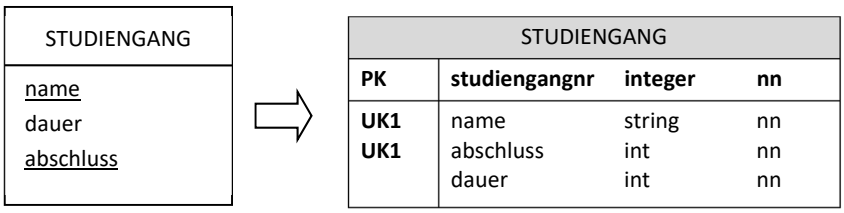
24 Ein solcher *künstlicher Schlüssel* wird auch als *surrogate key* bezeichnet.

Beispiel

Wendet man diese Regel auf den Entity-Typ `MODUL` aus der FH-Info-Fallstudie an, so ergibt sich folgendes Relationenschema:

MODUL			
PK	**modnr**	**string**	**nn**
	titel	string	nn
	workload	int	nn

Für den Entity-Typ `STUDIENGANG` wird ein künstliches Schlüsselattribut (`stgnr`) vom Typ `integer` hinzugefügt. Der ursprüngliche Schlüssel wird auf einen Unique-Key abgebildet[25]:



Da durch die Kombination (name, abschluss) ein weiterer (UK1) Schlüssel definiert wird, ist sichergestellt, dass es keine zwei Studiengänge mit identischen Werten für Bezeichnung und Abschluss geben kann.

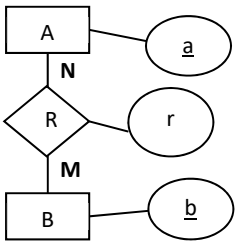
5.2 Transformation von Beziehungstypen

Da das Relationenmodell kein eigenständiges Beziehungskonzept bereitstellt, müssen Beziehungstypen mit Hilfe von Relationen, Schlüsseln und Fremdschlüssel abgebildet werden. Je nach Art der Beziehung sieht die Abbildung unterschiedlich aus. Für jede Art wird in den folgenden Abschnitten eine eigene Regel definiert.

5.2.1 N:M-Beziehungen

Die folgende Regel findet Anwendung bei binären N:M-Beziehungen und bei Beziehungen, für die keine Kardinalitäten definiert sind.

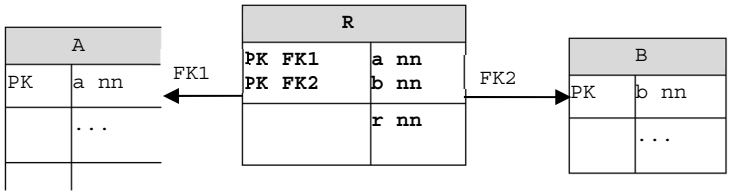
25 Um die Darstellung möglichst kompakt zu halten, wird in dieser und allen folgenden Abbildungen auf die Angabe von Datentypen verzichtet, falls diese keine Rolle im Zusammenhang mit der Transformation spielen.



Regel 2: Abbildung von N:M-Beziehungstypen

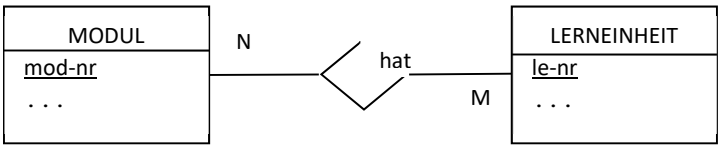
- Jeder Beziehungstyp *R* wird auf ein Relationenschema abgebildet. Dabei ist ein sinnvoller Name für dieses Schema zu vergeben.
- Alle Beziehungsattribute von *R* werden zu Attributen des neuen Schemas.
- Die Schlüsselattribute der Entity-Typen, die an der Beziehung *R* beteiligt sind, werden als zusätzliche Attribute dem neuen Schema hinzugefügt und als nicht optional gekennzeichnet.
- Die so hinzugefügten Attribute werden als Fremdschlüssel definiert und referenzieren die Relationen, die sich aus den an *R* beteiligten Entity-Typen ergeben haben.
- Der Primärschlüssel des neuen Relationenschemas setzt sich aus den hinzugefügten Schlüsselattributen zusammen.

D.h. für das dargestellte abstrakte ER-Fragment ergibt sich:

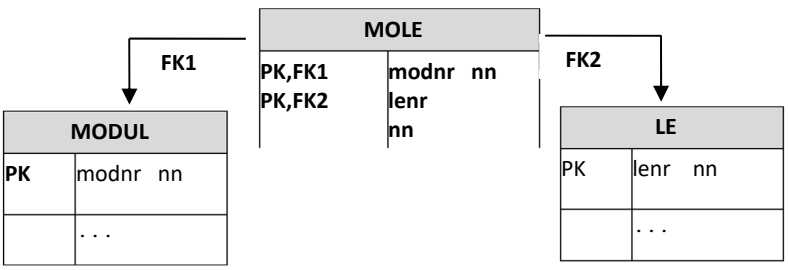


Beispiel:

In der FH-Info-Fallstudie besteht eine N:M-Beziehung zwischen MODUL und LERNEINHEIT:



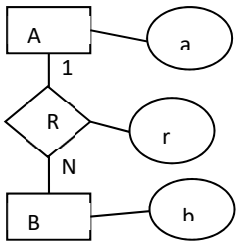
Für diese Beziehung `hat` wird eine neue Relation MOLE (**MO**DUL-hat-**LE**RNEINHEIT) gebildet:



Da die beiden Attribute modnr und lenr als Fremdschlüssel in die neue Relation MOLE aufgenommen wurden, ist sichergestellt, dass es zu jeder in MOLE gespeicherten Modulnummer bzw. zu jeder LE-Nummer auch einen Datensatz in der Relation MODUL bzw. LE gibt. Da beide Attribute zusammen den Primärschlüssel bilden, kann jede Modulnummer zusammen mit mehreren verschiedenen LE-Nummern gespeichert werden und umgekehrt.

5.2.2 1:N-Beziehungen

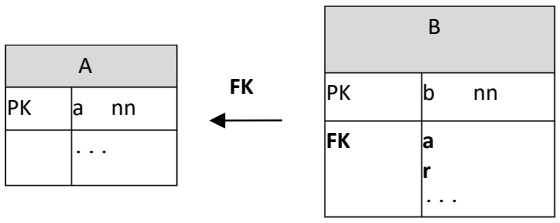
Bei der Abbildung von 1:N-Beziehungen kommt man im Vergleich zu den N:M-Beziehungen mit nur zwei Relationen aus. Das allgemeine Verfahren wird durch die folgende Regel definiert:



Regel 3: Abbildung von 1:N-Beziehungstypen
Die Beziehung *R* und die Relation, die aus dem Entity-Typ der N-Seite (B) entstanden ist, verschmelzen zu einer Relation.

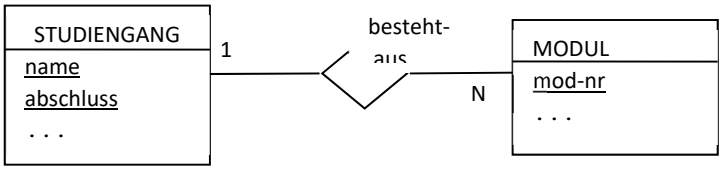
- Dieser Relation, wird der Schlüssel des Entity-Typs der 1-Seite (A) als optionales Fremdschlüsselattribut hinzugefügt.
- Alle Beziehungsattribute werden dieser Relation als optionale Attribute hinzugefügt.

D.h. für das dargestellte abstrakte ER-Fragment ergibt sich:

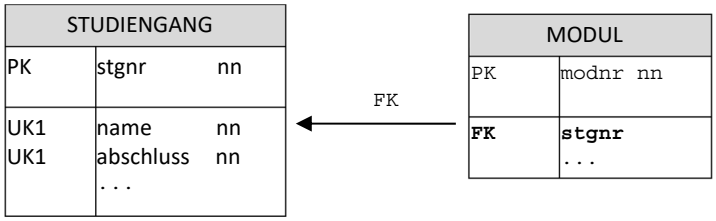


Beispiel:

In der FH-Info-Fallsatudie ist die Beziehung besteht-aus zwischen MODUL und STUDIENGANG ein Beispiel für den zweiten Fall einer 1:N-Beziehung:



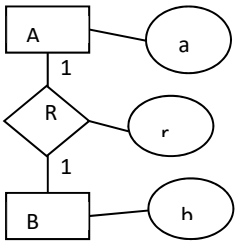
Die Beziehung besteht-aus und der Entity-Typ MODUL werden zu einer einzigen Relation MODUL. Da der Relation STG ein künstlicher Primärschlüssel hinzugefügt wurde (siehe Beispiel zu Kapitel 5.1), wird auch dieser als Fremdschlüsselattribut in die MODUL-Relation übertragen:



Das Fremdschlüsselattribut stgnr (Studiengangnr) in der Relation MODUL stellt sicher, dass ein Modul nur einem existierenden Studiengang zugeordnet werden kann.

5.2.3 1:1-Beziehungen

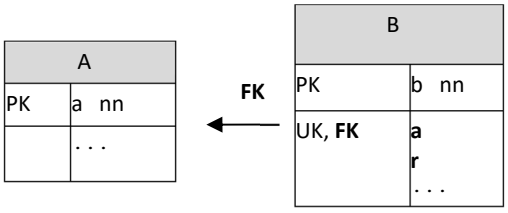
Bei der Abbildung von 1:1-Beziehungen kommt man ebenfalls mit nur zwei Relationen aus. Das allgemeine Verfahren wird durch die folgende Regel definiert:



Regel 4: Abbildung von 1:1-Beziehungstypen
Die Beziehung *R* und die Relation, die aus einem der beiden Entity-Typ entstanden ist, verschmelzen zu einer Relation.

- Dieser Relation wird der Schlüssel des Entity-Typs der anderen Seite als optionales Fremdschlüsselattribut hinzugefügt.
- Außerdem werden alle Beziehungsattribute zu optionalen Attributen dieser Relation.

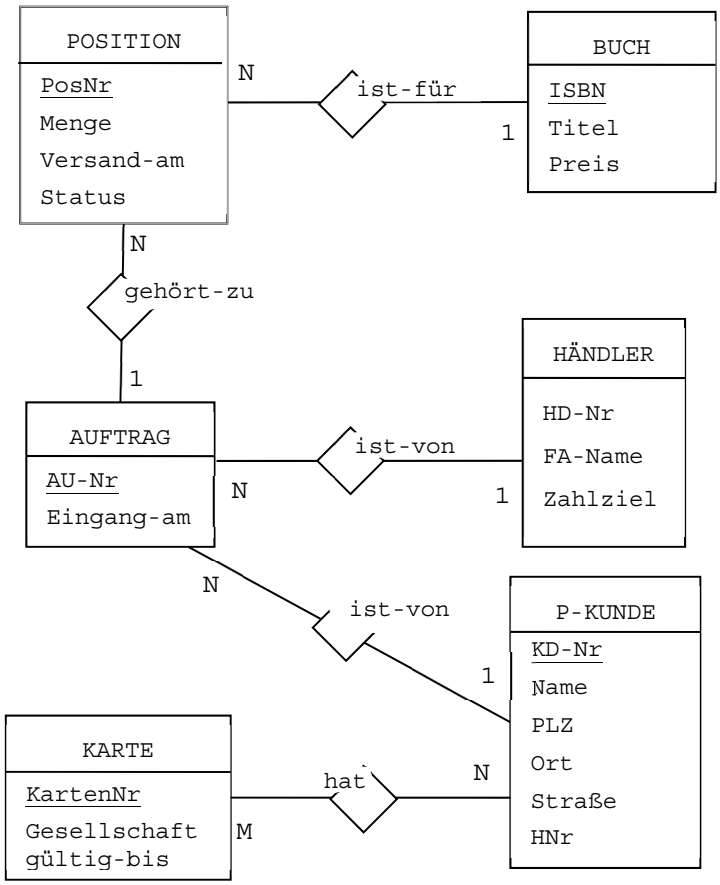
D.h. für das dargestellte abstrakte ER-Fragment ergibt sich:



5.3 Aufgaben

5.3.1 Fallstudie eLibri

Führen Sie für das abgebildete ER-Diagramm der Fallstudie *eLibri* die Schema-Transformation in das Relationen Modell durch.



6 Datenbankanfragen

Die Popularität des Relationen Modells und der Erfolg Relationaler Datenbanken liegt unter anderem in der Datenbanksprache SQL begründet. Der in der Praxis am häufigsten eingesetzte Teil der Datenbanksprache SQL dient zum Auswerten von Datenbanken. Eine Datenbankanfrage in SQL wird auch häufig als **`SELECT`-Anweisung** bezeichnet, da jede Datenbankanfrage mit dem Schlüsselwort `SELECT` beginnt. Die Beispielanfrage:

```
SELECT name, job, gehalt
  FROM ma
 WHERE abtnr = 20;
```

liefert zu den Mitarbeitern der Abteilung 20 den Namen, die Tätigkeit und das Grundgehalt und somit das folgende Ergebnis:

NAME	JOB	GEHALT
Meier	Projektleiter	3200
Karl	Manager	7200
Frohn	Programmierer	2900
Werther	Programmierer	2700
Meyer	Engineer	5200
Sorge	Engineer	5800

Jede SQL-Anfrage hat als Ergebnis eine Relation. Die Grundform der SELECT-Anweisung besteht aus drei Teilen:

```
SELECT Spaltenname1, ..., Spaltennamen
  FROM Tabellenname
 WHERE Bedingung
```

1. **`SELECT`**, bestimmt die Spalten der Ergebnisrelation[26].
2. **`FROM`**, legt die Basisrelation fest, aus der die Daten gelesen werden sollen.
3. **`WHERE`**, gibt die Bedingungen an, die von den Datensätzen der Basisrelationen zu erfüllen sind, damit sie zum Ergebnis gehören.

Der `WHERE`-Teil ist optional. Entfällt er, gehören alle Datensätze der Basisrelation zum Ergebnis.

Zum Verständnis dieses Kapitels ist es unbedingt erforderlich, die im Anhang befindliche Fallstudie BestTec zu lesen!

26 Um die Spaltenüberschriften von den Ergebnisdatensätzen optisch abzuheben, werden sie in diesem Kapitel immer in Großbuchstaben und Fettdruck dargestellt.

6.1 Syntax-Definition

In diesem und den folgenden Kapiteln wird die Sprache SQL vorgestellt. Dabei soll auf eine komplexe Systematik zur Sprachdefinition verzichtet werden. Anhand der SELECT-Anweisung soll das gewählte Vorgehen kurz erläutert werden:

```
SELECT Spaltenname1, ..., Spaltennamen
  FROM Tabellenname
 [WHERE Bedingung]
```

- Zur Syntaxdefinition wird die Schriftart `COURIER NEW` verwendet.
- Alle fettgedruckten und komplett großgeschriebenen Worte sind Schlüsselwort der Sprache SQL, z.B. **`SELECT.`**
- Alle kursiv geschriebenen Worte stehen für weitere Regeln, die entweder bereits definiert sind oder noch definiert werden. Falls *`Name`* in dem Wort auftaucht, handelt es sich um einen Bezeichner, der sich aus dem zugrunde liegenden Datenbankschema ableitet.
- Alle bei der Sprachdefinition in eckige Klammern `[]` gesetzten Teile sind nicht zwingend erforderlich und können weggelassen werden, z.B. **`[WHERE`** *`Bedingung`*`]` .

6.2 Werte und Datentypen

Datenbanken speichern Informationen und Daten in Form von Werten. Um konkrete Werte in SQL-Anweisungen aufzuschreiben, ist folgendes zu beachten:

Numerische Werte:
Numerische Werte werden als Folge von Ziffern geschrieben. Bei Festkommazahlen trennt der Dezimalpunkt die Vorkommastellen von den Nachkommstellen ab; z.B.:

```
1765   123.986      0.3243    .23665
```

Zeichenketten:
Zeichenketten sind beliebige Folgen von Ziffern, Buchstaben und Sonderzeichen, die in Anführungsstrichen (Hochkomma) eingeschlossen werden[27]; z.B.:

```
'Hello World'        '§234 Abs.7 BGB'
```

27 Die hier vorgestellte Syntax entspricht den Anforderungen des Datenbanksystems Oracle. Andere Datenbankhersteller verwenden statt einfachen Anführungsstrichen auch doppelte.

Datumswerte:
Für Datumswerte sieht der SQL-Standard eine Schreibweise vor, die sich an den jeweiligen nationalen Gegebenheiten [28] orientiert:

```
'20.02.2010'        '05.05.1999'
```

6.3 Projektion

Der SELECT-Teil bestimmt die Struktur (das Aussehen) der Ergebnisrelation. Dazu ist im Anschluss an das Schlüsselwort SELECT eine **Projektionsliste** anzugeben.

Wie bereits im einführenden Beispiel gezeigt, kann dies eine Liste von Attributen der Basistabellen sein. Im einfachsten Fall besteht die Projektionsliste aus einem *-Symbol. Durch dieses Symbol wird ausgedrückt, dass alle Spalten der Basistabelle ausgegeben werden sollen:

Alle Informationen zu den Projekten des Kunden M341.

```
SELECT *
  FROM prj
 WHERE kdnr = 'M341';
```

PRJNR	ABTNR	KDNR	TITEL
232	20	M341	Peak-Control

Die folgende Beispielanfrage

Name, Gehalt und die Prämie, die 110% eines Grundgehaltes ausmacht, für die Mitarbeiter der Abteilung 20.

```
SELECT name, gehalt AS Grundgehalt,
       gehalt*1.1 AS Prämie
   FROM ma
  WHERE abtnr = 20;
```

liefert als Ergebnis:

NAME	GRUNDGEHALT	PRÄMIE
Meier	3200	3520
Karl	7200	7920
Frohn	2900	3190
Werther	2700	2970
Meyer	5200	5720
Sorge	5800	6380

und illustriert zwei Aspekte des SELECT-Teils:

1. In der Projektionsliste können Werteausdrücke (z.B. `gehalt * 1.1`) auftauchen, die Ergebniswerte berechnen. Zur Formulierung

28 Die meisten Datenbanksysteme bieten die Möglichkeit, über Systemparameter die Internationalisierung des Systems zu steuern und damit nationale Besonderheiten bei Datumsangaben zu berücksichtigen.

von Wertausdrücken steht neben den Grundrechenarten eine Vielzahl von Operationen zur Verfügung, die hier nicht weiter betrachtet werden sollen.

2. Für jede Ergebnisspalte kann eine individuelle Überschrift festgelegt werden. Dies ist insbesondere für berechnete Spalten sinnvoll. Dazu gibt man nach dem Spaltenausdruck das Schlüsselwort **AS** gefolgt von einer Zeichenkette an, die Spaltenüberschrift bilden soll.

Duplikate eliminieren

Das Ergebnis einer Anfrage umfasst immer alle Datensätze, die die WHERE-Bedingung erfüllen. D.h. die folgende Anfrage

```
SELECT job
   FROM ma
  WHERE abtnr = 10;
```

⇨

JOB
Vertrieb
Geschäftsführer
Vertrieb
Controller
Vertrieb

Alle in der Abteilung 10 ausgeübten Tätigkeiten.

liefert insgesamt 5 Ergebnisdatensätze, von denen allerdings einige doppelt sind. Um Anfrageergebnisse von Duplikaten zu befreien, verwendet man **SELECT DISTINCT** anstatt SELECT:

```
SELECT DISTINCT job
   FROM ma
  WHERE abtnr = 10;
```

⇨

JOB
Vertrieb
Geschäftsführer
Controller

Die **verschiedenen** Tätigkeiten, die in Abteilung 10 ausgeübt werden!

Weitere Beispiele

Die Eliminierung von Duplikaten kann natürlich auch bei Projektionslisten vorgenommen werden, die mehr als ein Attribut enthalten:

```
SELECT DISTINCT job, abtnr
   FROM ma
  WHERE gehalt > 5000;
```

Die Tätigkeiten mit zugehörigen Abteilungen für die Mitarbeiter mit einem Gehalt über 5000.

JOB	ABTNR
Manager	50
Manager	20
Manager	40
Geschäftsführer	10
Engineer	20

SYNTAX

Der SELECT-Teil kann nach folgendem Muster aufgebaut werden:

```
SELECT [DISTINCT]
       Werteausdruck1 [AS Spaltenüberschrift1],
        ...
    Werteausdruckn  [AS Spaltenüberschriftn]
```

Ein ***Werteausdruck*** ist entweder ein Attributname oder eine Berechnung:

Syntax

Die elementarsten **Wertsaudrücke** sind:

- *Spaltenname*
- *Wert*

Werteausdrücke können geklammert werden:

- **(***Werteausdruck***)**

Werteausdrücke können Berechnungen darstellen

- *werteausdruck iOp werteausdruck*
  Mögliche Operatoren an Stelle von iOp sind: **+, -, *, /**

6.4 Selektion

Der **WHERE**-Teil dient zur Auswahl von Datensätzen. Dazu kann eine Bedingung formuliert werden. Alle Datensätze, für die diese Bedingung *wahr* ergibt, gehören zum Ergebnis der Anfrage:

```
SELECT *
  FROM prj
 WHERE kdnr = 'M341';
```

PRJNR	ABTNR	KDNR	TITEL
232	20	M341	Peak-Control

Die einfachsten Bedingungen lassen sich über den Vergleich von Attributen mit Wertausdrücken bilden. Dabei stehen die aus der Schulmathematik bekannten Vergleichsoperatoren

<	kleiner	**>**	größer	**=**	gleich
<=	kleiner gleich	**>=**	größer gleich	**<>**	ungleich

zur Verfügung. SQL kennt aber auch einige besondere Vergleichsoperatoren, die im Folgenden vorgestellt werden.

1. IN-Prädikat

Der Vergleich auf Gleichheit mit einer ganzen Wertemenge wird über das IN-Prädikat ausgedrückt:

Alle Mitarbeiter die Programmierer oder Trainer sind.

```
SELECT name, job, gehalt, abtnr
  FROM ma
 WHERE job IN ('Programmierer', 'Trainer');
```

NAME	JOB	GEHALT	ABTNR
Frohn	Programmierer	2900	20
Werther	Programmierer	2700	20
Beier	Trainer	2700	50
Klaas	Trainer	3200	50

An das IN-Prädikat muss sich eine Liste anschließen, in der Vergleichswerte für das vor dem IN stehendem Attribut aufgeführt sind. Alle Datensätze, für die das Attribut mit einem Wert aus der Liste übereinstimmt, erfüllen die Bedingung. Die Negation des IN-Prädikats wird durch die Kombination der beiden Wörter **NOT_IN** ausgedrückt. D.h. die folgende Anfrage würde alle Mitarbeiter liefern, die nicht zum Ergebnis des vorherigen Beispiels gehören:

Alle Mitarbeiter die weder Programmierer noch Trainer sind.

```
SELECT name, job, gehalt, deptno
FROM ma
  WHERE job NOT IN ('Programmierer', 'Trainer');
```

2. BETWEEN

Um zu überprüfen, ob ein Attribut einen Wert aus einem vorgegebenen Intervall besitzt, kann mit **BETWEEN** gearbeitet werden:

Mitarbeiter mit einem Gehalt zwischen 2800 und 3000.

```
SELECT name, gehalt
  FROM ma
WHERE gehalt BETWEEN 2800 AND 3000;
```

NAME	GEHALT
Frohn	2900
Schulte	2900
Dorn	3000
Holle	2900

Zusammen mit **BETWEEN** sind zwei Intervallgrenzen (erst Untergrenze, dann Obergrenze) anzugeben, die durch das Wort **AND** getrennt sind. Die Bedingung wird von allen Datensätzen erfüllt, deren Attributwert größer bzw. gleich der Untergrenze und kleiner bzw. gleich der Obergrenze sind.

3. LIKE

Über den Vergleichsoperator `LIKE` kann ein Attribut, in dem eine Zeichenkette gespeichert ist, mit einem Zeichenkettenmuster verglichen werden:

Alle Projekte, deren Titel mit der Buchstabenkombination `Control` endet.

```
SELECT *
  FROM prj
 WHERE titel LIKE '%Control';
```

PRJNR	ABTNR	KDNR	TITEL
232	20	M341	PeakControl
212	20	A12	HydroControl

Zur Bildung eines Vergleichsmusters stehen die Symbole `%` (Prozentzeichen) und `_` (Unterstrich) zur Verfügung.

`%`	Das Prozentzeichen steht als Platzhalter für eine beliebige Zeichenkette (die ggf. sogar leer sein kann).
`_`	Der Unterstrich steht für ein einzelnes, beliebiges Zeichen

D.h. bei dem Vergleich

```
name LIKE '_a%r_'
```

für das Attribut `name` aus der `MA`-Relation würden der Mitarbeiter `'Karl'` die Bedingung erfüllen, `'Langer'` würde sie nicht erfüllen.

6.4.1 Zusammengesetzte Bedingungen

Die meisten Anfragewünsche können nicht durch eine einzige Bedingung ausgedrückt werden. Will man alle Mitarbeiter sehen, die mehr als 2000 verdienen und in der Abteilung 20 arbeiten, sind bereits zwei Bedingungen zu formulieren. Um Bedingungen zu verknüpfen, bietet SQL die beiden Operationen **`AND`** und **`OR`**:

Alle Mitarbeiter der Abteilung 10 mit einem Gehalt, das über 4000 liegt[29].

```
SELECT name, gehalt, prov, abtnr
  FROM ma
 WHERE gehalt > 4000 AND abtnr = 10;
```

NAME	GEHALT	PROV	ABTNR
Langer	4800	1200	10
Hansen	9800		10
John	4900		10

29 `NULL`-Werte werden in den abgebildeten Ergebnistabellen durch leere Felder dargestellt.

Die beiden Operationen sind an die mathematische Logik angelehnt und entsprechen dem intuitiven Verständnis[30]:

`b1 AND b2`	Ergibt nur dann *wahr*, wenn beide Teilbedingungen wahr sind.
`b1 OR b2`	Ergibt *wahr*, wenn mindestens eine der Teilbedingungen wahr ist.

Grundsätzlich lassen sich natürlich Bedingungen formulieren, in denen AND und OR gemeinsam auftreten. Dabei ist zu beachten, dass AND stärker als OR bindet, also AND vor OR ausgeführt wird. Somit gehört bei der folgenden Anfrage

```
SELECT name, gehalt, job, abtnr
FROM ma
WHERE gehalt > 5500
  AND job = 'Manager'
   OR job = 'Engineer';
```

jeder Mitarbeiter zum Ergebnis, der eine der beiden folgenden Bedingungen erfüllt ist:

- er ist Manager und verdient mehr als 5500
- er ist Engineer.

Durch das Setzen von Klammern kann die Auswertung einer Bedingung beeinflusst werden. Um im Ergebnis nur Mitarbeiter zu erhalten, die mehr als 5500 verdienen und dabei einen der beiden Jobs (Engineer oder Berater) ausübt, müssen die folgenden Klammern hinzugefügt werden:

```
WHERE gehalt > 5500
  AND (tätigkeit = 'Manager' OR
       tätigkeit = 'Engineer' );
```

Neben AND und OR kennt SQL mit NOT noch einen dritten Operator, der als Präfix vor eine Bedingung geschrieben werden kann:

```
WHERE NOT tätigkeit = 'MANAGER'
```

NOT steht für die Negation. Da NOT stärker bindet als AND, wird NOT immer vor AND ausgewertet[31].

30 b1 und b2 stehen hier für zwei beliebige Bedingungen, die *wahr* oder *falsch* ergeben können.

31 In der Praxis braucht man das NOT nur sehr selten, da man im Allgemeinen statt einer Bedingungen der Art `NOT tätigkeit = 'MANAGER'` immer `tätigkeit <> 'MANAGER'` schreiben würde.

6.4.2 Syntax der `WHERE`-Bedingung

Bedingungen können durch den Wertevergleich gebildet werden:

- *Spaltenname vOp Werteausdruck*
  Mögliche Vergleichsoperatoren an Stelle von vOp sind:
  `=, <=, >=, <, >, <>`

Über **`IN`** bzw. **`NOT IN`** kann ein Attribut mit einer Werteliste vergleichen werden:

- *Spaltenname* ***[NOT] IN*** *(*$wert_1$*,..,* $wert_n$*)*

Mit **`BETWEEN`** wird geprüft, ob ein Attribut zwischen zwei Intervallgrenzen liegt :

- *Spaltenname* ***BETWEEN*** $wert_1$ ***AND*** $wert_2$

Ein textwertiges Attribut kann über LIKE mit einem Zeichenkettenmuster verglichen werden:

- *Spaltenname* ***LIKE*** *Zeichenkettenmuster*

Bedingungen können geklammert werden:

- ***(****Bedingung****)***

Zwei Bedingungen können mit AND bzw. OR verknüpft werden

- $Bedingung_1$ ***AND*** $Bedingung_2$
- $Bedingung_1$ ***OR*** $Bedingung_2$

Über den Operator **`NOT`** kann eine Bedingung negiert werden

- ***NOT*** *Bedingung*

6.4.3 Aufgaben

Formulieren Sie zu jeder der folgenden Aufgaben eine SQL-Anfrage, die das gewünschte Ergebnis liefert! Die Struktur der Ergebnistabelle ist jeweils unter der Aufgabenstellung abgebildet.

1. Bestimmen Sie alle Mitarbeiter, deren Grundgehalt zwischen 2500.- und 4000.- liegt. Dabei sollen auch Mitarbeiter im Ergebnis erscheinen, die genau 2500.- oder genau 4000.- verdienen.

NAME	GEHALT

2. Modifizieren Sie Lösung der Aufgabe 1. so, dass jetzt die Mitarbeiter, die 2500.- oder 4000.- verdienen nicht mehr mit im Ergebnis auftauchen.

3. Bestimmen Sie alle Mitarbeiter der Abteilungen 10 oder 20.

NAME	GEHALT	ABTNR

4. Bestimmen Sie alle Mitarbeiter, deren Name 4 Buchstaben enthält und dabei an der zweiten Stelle den Buchstaben „o“!

NAME	ABTNR

5. Bestimmen Sie alle Mitarbeiter der Abteilungen 10 oder 20, deren ein Grundgehalt zwischen 2500.- und 4000.- liegt (inkl. 2500.- und 4000.-).

NAME	GEHALT	ABTNR

6. Welche Mitarbeiter der Abteilungen 20, 30 oder 40 haben einen Namen, der 4 Buchstaben enthält und dabei an der zweiten Stelle den Buchstaben „o“?

NAME	ABTNR

7. Gesucht sind Mitarbeiter der Abteilungen 10 oder 20. Dabei sollen nur Mitarbeiter ausgegeben werden, die Manager sind oder ein Grundgehalt haben, das über 5000 liegt.

NAME	ABTNR	GEHALT

8. Gesucht sind alle Mitarbeiter, die einem der folgenden beiden Profile entsprechen:

   - Das Grundgehalt ist kleiner als 5000 und die Tätigkeit ist weder Programmierer noch Administrator
   - Die Tätigkeit ist Controller oder Trainer.

NAME	ABTNR	JOB	GEHALT

6.5 Sortieren

Die Reihenfolge der Ergebnisdatensätze ist im Prinzip zufällig und kann bei der wiederholten Ausführung einer Anfrage unterschiedlich ausfallen. Durch den Zusatz ORDER BY am Ende einer Anfrage kann die Sortierung der Ergebnisdatensätze festgelegt werden:

```
SELECT name, job, gehalt, abtnr
  FROM ma
 WHERE abtnr = 10
ORDER BY gehalt;
```

Ausgabe aller Mitarbeiter der Abteilung 10 sortiert nach dem Grundgehalt.

NAME	JOB	GEHALT	ABTNR
Schulte	Vertrieb	2900	10
Kurt	Vertrieb	3100	10
Langer	Vertrieb	4800	10
John	Controller	4900	10
Hansen	Geschäftsführer	9800	10

Durch den Zusatz **ASC** (für aufsteigend) oder **DESC** (für absteigend) kann die Sortierreihenfolge bestimmt werden[32]. Es können grundsätzlich mehrere Sortierkriterien angegeben werden, wobei für jedes einzelne die Reihenfolge individuell bestimmt werden kann. Als Sortierkriterium ist ein beliebiger Werteausdruck zulässig:

Ausgabe von Mitarbeitern sortiert nach Tätigkeit und Höhe des Weihnachtsgeldes.

```
SELECT name, job, gehalt*1.2 AS WGeld
  FROM ma
 WHERE abtnr = 10
ORDER BY job DESC, gehalt*1.2 ASC;
```

NAME	JOB	WGeld
Schulte	Vertrieb	3480
Kurt	Vertrieb	3720
Langer	Vertrieb	5760
Hansen	Geschäftsführer	11760
John	Controller	5880

Falls mehrere Sortierkriterien angegeben sind, wird zunächst das als erstes angegebene Kriterium (hier job DESC) verwendet. Bei Datensätzen, die in diesem Kriterium gleiche Werte aufweisen, wird das zweite Sortierkriterium angewandt. Falls auch dort noch gleiche Werte auftreten, kommt das nächste Kriterium (falls vorhanden) zur Anwendung, usw.

Syntax

Die Sortierung wird durch den `ORDER BY` Teil bestimmt:

```
ORDER BY werteausdruck1 [ASC|DESC],
             ...
         werteausdruckn [ASC|DESC]
```

32 `ASC` steht für Ascending und `DESC` für Descending

6.5.1 Aufgaben

Modifizieren Sie die Lösungen der vorhergehenden Aufgaben, so dass die Ergebnistabellen wie folgt sortiert werden:

9. Die Lösung zu Aufgabe 5 soll zunächst nach Abteilungsnummern aufsteigend sortiert werden und dann innerhalb gleicher Abteilungen absteigen nach dem Gehalt.

10. Die Lösung zu Aufgabe 8 soll zunächst nach Job, dann nach Abteilungsnummer und dann nach Gehalt sortiert werden. Die Sortierung soll grundsätzlich aufsteigend erfolgen.

6.6 Verbundbildung

In der Praxis lässt sich ein Informationsbedarf in den meisten Fällen nicht auf der Basis einer einzigen Relation befriedigen. Will man z.B. für jeden Mitarbeiter nicht nur den Namen und das Gehalt, sondern auch noch den Sitz seiner Abteilung bestimmen, so benötigt man neben der MA-Relation auch noch die ABT-Relation. Dies ist sowohl für den menschlichen Betrachter der Daten als auch für die Datenbank der Fall. Als Betrachter, entnimmt man für jeden Mitarbeiter die Abteilungsnummer der MA-Tabelle und sucht mit dieser Nummer in der ABT-Tabelle nach einem passenden ABT-Datensatz. Dort findet sich dann der Sitz der zugehörigen Abteilung.

Um einen solchen Anfragewunsch in SQL zu formulieren, muss man in dem FROM-Teil mehrere Basistabellen angeben[33]:

Ausgabe aller Mitarbeiter, die als Engineer tätig sind, mit ihrem Dienstsitz.

```
SELECT ma.manr, ma.name, ma.job, abt.sitz
  FROM ma, abt
 WHERE ma.abtnr = abt.abtnr
   AND ma.job = 'Engineer';
```

MANR	NAME	JOB	SITZ
8956	Dorn	Engineer	Bielefeld
8322	Frei	Engineer	Bielefeld
8247	Holle	Engineer	Paderborn
7769	Meyer	Engineer	Verl
8659	Sorge	Engineer	Verl
8969	Walter	Engineer	Paderborn

Die Anfrage liefert aber nur dann ein sinnvolles Ergebnis, wenn man im WHERE-Teil eine Verknüpfungsbedingung für die Datensätze der beiden Tabellen angibt34. Unterbleibt die Bedingung, so verknüpft die Datenbank jeden Datensatz der ersten Relation mit jedem Datensatz der zweiten Relation35. D.h. es entsteht eine Vielzahl völlig unsinniger Datensatzkombinationen.

Anfragen, die mehr als eine Basisrelation umfassen, werden als Verbundanfragen (in der Datenbankwelt wird der Verbund auch Join genannt) bezeichnet.

Im Unterschied zu den Anfragen der ersten beiden Teilkapitel stellen wir bei einer Verbundanfrage jedem Spaltennamen den Tabellennamen (`ma.name` statt nur `name`) voran, um eine eindeutige Zuordnung

33 Dies sind genau die Relationen, die der menschliche Betrachter auch zur Bestimmung des Ergebnisses‘ verwenden würde.

34 Diese Bedingung entspricht wiederum der Verknüpfung, die auch der menschliche Betrachter vornimmt, wenn er von einer Tabelle zu einer anderen Tabelle blickt.

35 Dies ist mathematisch das Kreuzprodukt zwischen den beiden beteiligten Relationen. Im konkreten Fall würde dies bedeuten, dass 100 Paarung (20 MA-Datensätze x 5 ABT-Datensätze) entstehen.

der Spalten zu den Tabellen zu gewährleisten. Dort, wo eine Spalte aus nur einer der im FROM-Teil verwendeten Tabellen stammen kann, darf die Angabe des vorangestellten Tabellennamens auch unterbleiben. D.h. die auf der vorhergehenden Seite abgebildete Beispielanfrage hätte auch wie folgt geschrieben werden können:

```
SELECT manr, name, job, sitz
  FROM ma, abt
 WHERE ma.abtnr = abt.abtnr
   AND job = 'Engineer';
```

Syntax

Der FROM-Teil bestimmt die Basis-Relationen einer Anfrage:

```
FROM Tabellenname1 ,..., Tabellennamen
```

6.6.1 Aufgaben

11. Bestimmen Sie alle Mitarbeiter die in der Abteilung aus Münster tätig sind.

ABTNR	BEZEICHNUNG	MANR	NAME

12. Bestimmen Sie alle Mitarbeiter die in der Abteilung Education oder Zentrale tätig sind.

ABTNR	BEZEICHNUNG	MANR	NAME

13. Bestimmen Sie alle Mitarbeiter die in der Abteilung Education oder Zentrale tätig sind. Gesucht werden nur Mitarbeiter deren Gehalt über 3000 liegt. Das Ergebnis ist nach Abteilungsbezeichnung aufsteigend zu sortieren. Innerhalb einer Abteilung soll die Sortierung absteigend nach dem Gehalt erfolgen.

ABTNR	BEZEICHNUNG	NAME	GEHALT

6.7 Daten konsolidieren

Mit den bisher eingeführten Sprachmitteln lassen sich Anfragen nach dem Durchschnittsgehalt aller Mitarbeiter oder der Anzahl Mitarbeiter je Abteilung nicht formulieren. In der Praxis sind aber gerade solche Anfragen häufig von besonderem Interesse. Hierzu müssen Werte über mehrere Datensätze hinweg konsolidiert bzw. aggregiert werden. Dies geschieht in SQL mit Hilfe sogenannter Gruppenfunktionen.

6.7.1 Gruppenfunktionen

SQL stellt fünf Gruppenfunktionen zur Verfügung. Soll das Durchschnittsgehalt der Mitarbeiter der Abteilung 10 bestimmt werden, geschieht dies mit der AVG-Funktion[36]:

Das Durchschnittsgehalt aller Mitarbeiter.

```
SELECT AVG(gehalt) AS AVG_GH
  FROM ma
 WHERE abtnr = 10;
```

⇨

AVG_GH
5100

Bei der Auswertung der Anfrage bestimmt die Datenbank zunächst alle MA-Datensätze, der Abteilung 10 und berechnet dann das arithmetische Mittel die Werte des Gehalt-Attributs dieser Datensätze.
Die fünf Gruppenfunktionen aus dem SQL-Kern sind in der folgenden Tabelle zusammengefasst:

Gruppenfunktion	bestimmt …
AVG(...)	… das arithmetische Mittel
SUM(...)	… die Summe
MIN(...)	… den kleinsten Wert
MAX(...)	… den größten Wert
COUNT(...)	… die Anzahl

Alle Gruppenfunktionen benötigen als Parameter einen Werteausdruck. In den meisten Fällen ist dies allerdings ein Attribut wie in dem einführenden Beispiel.

36 AVG steht als Abkürzung für *average*

COUNT(*)

Eine Ausnahme bildet COUNT. Hier ist auch folgende Konstruktion erlaubt:

Die Anzahl Mitarbeiter mit einem Gehalt kleiner 3000.

```
SELECT COUNT(*) AS Anzahl
  FROM ma
 WHERE gehalt < 3000;
```

⇨

Anzahl
7

Durch COUNT(*) wird die Anzahl Datensätze ermittelt, die die angegebene WHERE-Bedingung erfüllen.

Duplikate

Die Gruppenfunktionen haben die Eigenart, dass sie grundsätzlich Duplikate **nicht** ausblenden. D.h. die Anfrage

```
SELECT COUNT(job) AS Anzahl
  FROM ma;
```

⇨

Anzahl
20

liefert als Ergebnis den Wert 20 und nicht 9. Sollen Duplikate ausgeblendet werden, so muss vor das Argument der Gruppenfunktion **DISITNCT** geschrieben werden[37]:

```
SELECT COUNT(DISTINCT job) AS Anzahl
  FROM ma;
```

Weitere Beispiele

Grundsätzlich können mehrere Gruppenfunktionen in einer Projektionsliste verwendet werden:

Die Anzahl, das niedrigste und das höchste Grundgehalt und der Durchschnitt des Weihnachtsgelds der Mitarbeiter der Abteilung 20.

```
SELECT COUNT(manr) AS ANZ_MA,
       MIN(gehalt) AS MIN_GH,
       AVG(gehalt*1.2) AS AVG_WGeld,
       MAX(gehalt) AS MAX_GH
  FROM ma
 WHERE abtnr = 20;
```

ANZ_MA	MIN_GH	AVG_WGeld	MAX_GH
6	2700	5400	7200

Syntax

Ausdrücke mit Gruppenfunktionen werden wie folgt gebildet:

- `Gruppenfunktion([DISTINCT] Werteausdruck)`
  `Gruppenfunktionen sind: AVG, SUM, COUNT, MIN, MAX`
- **COUNT (*)**

37 DISITNCT macht nur in der Gruppenfunktion COUNT Sinn.

6.7.2 Aufgaben

14. Bestimmen Sie die Anzahl aller Mitarbeiter.

ANZAHL

15. Bestimmen Sie die Anzahl der Mitarbeiter der Abteilungen 30 und 40.

ANZAHL

16. Bestimmen Sie die Anzahl der Mitarbeiter, die in Gütersloh oder Bielefeld tätig sind.

ANZAHL

17. Bestimmen Sie die Anzahl der Mitarbeiter, das Durchschnittsgehalt und die Summe der Gehälter aller Mitarbeiter, die in Gütersloh oder Bielefeld tätig sind.

ANZAHL	D_GEHALT	SUMME_GEHALT

6.7.3 Gruppenbildung

Soll die Anzahl Mitarbeiter je Tätigkeit bestimmt werden, so muss die Ergebnistabelle zwei Spalten aufweisen: die Tätigkeitsbezeichnung und die zugehörige Anzahl Mitarbeiter. Die folgende Anfrage liefert das gewünschte Ergebnis:

```
SELECT job, COUNT(manr) AS AnzMA
  FROM ma
 WHERE gehalt > 4500
 GROUP BY job;
```

JOB	AnzMA
Manager	4
Geschäftsführer	1
Vertrieb	1
Engineer	2
Controller	1

Das Beispiel zeigt eine Besonderheit im Umgang mit Gruppenfunktionen. Immer wenn in der Projektionsliste Gruppenfunktionen und Attribute in getrennten Spalten nebeneinander ausgegeben werden, muss

ein GROUP BY eingefügt werden, in dem die außerhalb der Gruppenfunktionen stehenden Attribute aus der Projektionsliste aufgezählt sind.
Durch GROUP BY werden die Datensätze, die die WHERE-Bedingung erfüllen, in Gruppen eingeteilt, so dass alle Datensätze, die die gleichen Werte für die Attribute des GROUP BY Teils aufweisen, in eine Gruppe gelangen. Bei der Ausgabe der Ergebnistabelle wird die Projektionsliste für jede Gruppe genau einmal ausgewertet.
D.h. in dem obigen Beispiel werden alle Mitarbeiter bestimmt, die mehr als 4500 verdienen. Diese werden dann je Tätigkeit (also je vorhandenem Wert des Attributes job) in Gruppen eingeteilt. Für jede Gruppe wird die Tätigkeit und die Anzahl Mitarbeiter ausgegeben.

Weitere Beispiele

1. Natürlich kann es mehr als ein Gruppierungskriterium in einer Anfrage geben:

```
SELECT abtnr, COUNT(manr) AS AnzMA, job
  FROM ma
 WHERE abtnr in (30, 40, 50)
 GROUP BY job, abtnr;
```

Die Anzahl Mitarbeiter je Tätigkeit und Abteilung.

ABTNR	AnzMA	JOB
30	2	Engineer
30	1	Manager
40	2	Engineer
40	1	Manager
50	1	Manager
50	2	Trainer

Die Reihenfolge der Werteausdrücke in der Projektionsliste und die Reihenfolge der Gruppierungskriterien im GROUP BY spielen keine Rolle.

2. Unabhängig von dem gewählten Gruppierungskriterium können auch mehrere Gruppenfunktionen in der Projektionsliste verwendet werden.

```
SELECT ma.abtnr, sitz, COUNT(manr) AS AnzMA,
       AVG(gehalt) AS AVG_GH,
       MIN(gehalt) AS MIN_GH
  FROM ma, abt
 WHERE ma.abtnr = abt.abtnr
 GROUP BY ma.abtnr, sitz;
```

Die Anzahl Mitarbeiter, das Durchschnittsgehalt und das kleinste Gehalt je Abteilung.

ABTNR	SITZ	AnzMA	AVG_GH	MIN_GH
10	Gütersloh	5	5100,00	2900,00
20	Verl	6	4500,00	2700,00
30	Paderborn	3	3400,00	2500,00
40	Bielefeld	3	3666,67	2500,00
50	Münster	3	3700,00	2700,00

Wie dieses Beispiel zeigt, kann natürlich auch die bereits bekannt Verbundbildung eingesetzt werden[38].

Syntax

```
GROUP BY  [Tabellenname1.]Spaltenname1,
                  ...,
          [Tabellennamen.] Spaltennamen
```

6.7.4 Aufgaben

18. Bestimmen Sie die Anzahl der Mitarbeiter je Abteilungsnummer.

ABTNR	ANZAHL

19. Bestimmen Sie die Anzahl der Mitarbeiter je Abteilungsbezeichnung.

BEZEICHNUNG	ANZAHL

20. Bestimmen Sie Durchschnittsgehalt je Abteilung für die Abteilungen mit Sitz in Gütersloh, Bielefeld oder Paderborn.

ABTNR	BEZEICHNUNG	D_GEHALT

21. Bestimmen Sie je Job die Anzahl der Mitarbeiter und das Durchschnittsgehalt. Das Ergebnis soll nach dem Durchschnittsgehalt aufsteigend sortiert sein.

JOB	ANZAHL	D_GEHALT

38 Bei der Verwendung von Tupelvariablen im `GROUP-BY`-Teil müssen diese identisch mit denen in der Projektionsliste sein.

6.7.5 Gruppenauswahl

In dem WHERE-Teil einer Anfrage werden die Datensätze ausgewählt, die zum Ergebnis gehören. Mit der Gruppenbildung besteht jetzt neben der Auswahl von Datensätzen zusätzlich der Bedarf für Bedingungen zur Gruppenauswahl, um z.B. alle Abteilungen zu bestimmen, in denen das Durchschnittsgehalt größer 4000 ist. Für die Gruppenauswahl muss eine SELECT-Anfrage um einen HAVING-Teil erweitert werden:

```
SELECT abtnr, AVG(gehalt) AS AVG_GH
  FROM ma
 GROUP BY abtnr
 HAVING AVG(gehalt) > 4000;
```

ABTNR	AVG_GH
10	5100,00
20	4500,00

Die Bedingung im HAVING-Teil dient zur Gruppenauswahl und basiert daher auf den zuvor eingeführten Gruppenfunktionen. Eine Bedingung zur Zeilenauswahl (z.B. `job = 'MANAGER'`) ist im HAVING-Teil nicht zulässig[39].

Natürlich können auch im HAVING-Teil wieder mit AND und OR Teilbedingungen zu größeren Bedingungen zusammengesetzt werden. Dabei ist es nicht erforderlich, dass die im HAVING-Teil mit Gruppenfunktionen bestimmten Werte auch im SELECT-Teil ausgegeben werden, wie das folgende Beispiel zeigt:

```
SELECT abtnr, AVG(gehalt) AS AVG_GH
  FROM ma
 GROUP BY abtnr
 HAVING AVG(gehalt) > 3500
    AND COUNT(manr) < 5;
```

Falls eine Abteilung weniger als 5 Mitarbeiter hat und das Durchschnittsgehalt größer 3500 ist, werden die Nummer und das Durchschnittsgehalt der Abteilung ausgegeben.

ABTNR	AVG_GH
50	3700,00
40	3666,67

Weitere Beispiele

Die Verwendung von GROUP-BY und HAVING schränken den Einsatz des WHERE-Teils nicht ein. Grundsätzlich erfolgt vor der Gruppenbildung und der Gruppenauswahl eine Zeilenauswahl durch die WHERE-Bedingung:

```
SELECT job, AVG(gehalt) AS AVG_GH
     FROM ma
    WHERE abtnr in (10, 20)
```

Tätigkeiten bei denen das Durchschnittsgehalt in den Abteilungen 10 und 20 über 4000 liegt.

39 Genauso ist eine Bedingung (z.B. `AVG(gehalt)>3000`), die auf einer Gruppenfunktion basiert, im WHERE-Teil nicht zulässig!

```
GROUP BY job
HAVING AVG(gehalt) > 4000;
```

Job	AVG_GH
Manager	7200,00
Geschäftsführer	9800,00
Engineer	5500,00
Controller	4900,00

Syntax

```
HAVING bedingung
```

Die Syntax zur Bildung von Bedingungen wurde bereits im Zusammenhang mit dem WHERE-Teil eingeführt. Für die Bedingung im HAVING-Teil gilt, dass sie sich auf die Gruppierungsattribute aus dem GROUP BY-Teil beziehen darf und beliebige Werteausdrücke enthalten darf, die mittels Gruppenfunktionen gebildet werden.

6.7.6 Aufgaben

22. Bestimmen Sie die Anzahl der Mitarbeiter je Abteilung. Es sollen nur Abteilungen ausgegeben werden, in den weniger als 5 Personen tätig sind.

BEZEICHNUNG	ANZAHL

23. Bestimmen Sie je Job die Anzahl der Mitarbeiter und das Durchschnittsgehalt. Das Ergebnis soll nach dem Durchschnittsgehalt aufsteigend sortiert sein. Es sollen nur Abteilungen ausgegeben werden, in denen mehr als 4 Mitarbeiter tätig sind oder bei denen das Durchschnittsgehalt über 4000.- liegt!

JOB	ANZAHL	D_GEHALT

6.8 Unteranfragen

Zur Formulierung von Bedingungen wurde bereits das IN-Prädikat in Abschnitt 6.4 vorgestellt:

```
SELECT name, job, gehalt, abtnr
    FROM ma
  WHERE job IN ('Programmierer', 'Trainer');
```

Die zum Vergleich verwendete Wertemenge kann nicht nur durch Aufzählen sondern auch durch eine (Unter-) Anfrage bestimmt werden:

Alle Abteilungen, in denen Mitarbeiter tätig sind, die mehr als 6000 verdienen.

```
SELECT abtnr, bezeichnung, sitz
  FROM abt
WHERE abtnr IN (SELECT abtnr
                  FROM ma
                 WHERE gehalt > 6000);
```

ABTNR	BEZEICHNUNG	SITZ
10	Zentrale	Gütersloh
20	Engineering	Verl

Natürlich kann genauso mit NOT IN gearbeitet werden, um eine Unteranfrage im WHERE-Teil einzubinden:

Alle Abteilungen, in denen kein Projekt stattfindet.

```
SELECT abtnr, bezeichnung, sitz
  FROM abt
 WHERE abtnr NOT IN (SELECT abtnr
                       FROM prj);
```

ABTNR	BEZEICHNUNG	SITZ
30	Automotive	Paderborn
40	Robotik	Bielefeld
50	Education	Münster

Von einer Unteranfrage spricht man, wenn eine Anfrage als Teil innerhalb einer anderen umfassenderen Anfrage verwendet wird. Diese umfassende Anfrage wird auch als äußere Anfrage bezeichnet. Es ist möglich, Anfragen beliebig tief ineinander zu verschachteln.

Einzelwertige Unteranfragen

Der direkte Vergleich[40] eines Attributes mit dem Ergebnis einer Unteranfrage ist immer dann möglich, wenn die Unteranfrage nur einen Wert zum Ergebnis hat:

Alle Mitarbeiter, deren Gehalt über dem Durchschnittsgehalt der Abteilung 20 liegt.

```
SELECT manr, name, gehalt
  FROM ma
```

40 Unter Verwendung eines der Vergleichsoperatoren: <, >, <=, >=, =, <>

```
WHERE gehalt > (SELECT AVG(gehalt)
                  FROM ma
                 WHERE abtnr = 20);
```

MANR	Name	Gehalt
7900	Karl	7200
8801	Langer	4800
7656	Hansen	9800
7767	John	4900
...	...	...

Durch die Verwendung einer Gruppenfunktion in der Unteranfrage ist sichergestellt, dass die Unteranfrage nur einen einzigen Wert liefert und der Vergleich mit dem Attribut `gehalt` Sinn macht.

Unteranfragen im HAVING-Teil

Unteranfragen können nicht nur im `WHERE`-Teil sondern auch im `HAVING`-Teil verwendet werden:

Alle Abteilungen deren Durchschnittsgehalt unterhalb des der Abteilung 20 liegt.

```
SELECT abtnr, avg(gehalt)
  FROM ma
 GROUP BY abtnr
HAVING AVG(gehalt) < (SELECT AVG(gehalt)
                        FROM ma
                       WHERE abtnr = 20);
```

ABTNR	AVG_GH
30	3400,00
50	3700,00
40	3666,67

Syntax

Unteranfragen erweitern die syntaktischen Möglichkeiten zur Formulierung von **Bedingungen**:

Bedingungen mit Unteranfragen lassen sich im `WHERE`- und im `HAVING`-Teil wie folgt formulieren:

- `Werteausdruck` **`[NOT] IN`** `(Select-Anfrage)`
- `Werteausdruck vOp (Select-Anfrage)`
  Mögliche Vergleichsoperatoren an Stelle von vOp sind:
  `=, <=, >=, <, >, <>.`

SQL kennt eine Reihe weiterer Operationen, um Unteranfragen in den `WHERE`- oder `HAVING`-Teil zu integrieren. Auf die Vorstellung wird verzichtet, da die Ausdruckmächtigkeit der Sprache sich hierdurch nicht erweitert.

6.8.1 Aufgaben

24. Bestimmen Sie die Projekte der Abteilung, die ihren Sitz in Gütersloh hat. Lösen Sie das Problem auf zwei Wegen:

    a. Unter Verwendung einer Unteranfrage.

    b. Mit Hilfe einer Verbundbildung.

KDNr	TITEL

25. Bestimmen Sie alle Abteilungen in denen kein Engineer tätig ist.

ABTNR	BEZEICHNUNG

26. Welche Mitarbeiter verdienen mehr als das Durchschnittsgehalt aller Mitarbeiter?

MANR		NAME	GEHALT

27. Bestimmen Sie die Anzahl der Mitarbeiter je Abteilung. Es sollen nur Abteilungen ausgegeben werden, in denen mehr Personen tätig sind als in der Abteilung 10.

BEZEICHNUNG	ANZAHL

28. Bestimmen Sie das Durchschnittsgehalt der Mitarbeiter je Abteilung. Es sollen nur Abteilungen ausgegeben werden, in denen mehr Personen tätig sind als in der Abteilung mit Sitz in Münster.

BEZEICHNUNG	D_GEHALT

6.9 SQL-Anfragen - Zusammenfassung

Über die SELECT-Anweisung lassen sich in SQL Anfragen an Datenbanken formulieren. Die dafür vorhandenen Sprachmittel wurden in diesem und dem vorhergehenden Teilkapitel erläutert und an einer Vielzahl von Beispielen illustriert. Über den hier vorgestellten Sprachumfang hinaus, bietet SQL noch einige weitere Möglichkeiten SELECT-Anfragen zu formulieren. Die meisten der nicht näher betrachteten Sprachelemente erhöhen allerdings die Ausdrucksmächtigkeit nicht, sondern erlauben nur alternative Formulierungen (wie es z.B. auch bei den Verbundausdrücken der Fall ist.) Grundsätzlich kann eine SELECT-Anfrage aus den folgenden sechs Teilen bestehen:

SELECT	Bestimmt in einer Projektionsliste die Struktur der Ergebnistabelle. Neben Werteausdrücken können Gruppenfunktionen zum Einsatz kommen.
FROM	Gibt die Basistabellen an, die in der Anfrage verwendet werden.
WHERE	Legt in einer Bedingung fest, welche Datensätze zum Ergebnis gehören. Durch die Verwendung von AND und OR können komplexe Bedingungen formuliert werden. Innerhalb des WHERE-Teils können wiederum SELECT-Anfragen als Unteranfragen auftauchen.
GROUP BY	Bestimmt die Gruppierungskriterien. Dieser Teil wird immer benötigt, wenn Gruppenfunktionen und Werteausdrücke in der Projektionsliste gemeinsam auftreten.
HAVING	Dient zur Gruppenauswahl. Genau wie im WHERE-Teil können AND und OR verwendet werden und Unteranfragen auftauchen.
ORDER BY	Legt die Reihenfolge der Datensätze in der Ergebnistabelle fest.

Die ersten beiden Elemente (SELECT und FROM) müssen grundsätzlich in einer Anfrage auftauchen, alle folgenden Teile können im Prinzip entfallen.

Die intuitive Ausführungsreihenfolge einer Anfrage sieht wie folgt aus:

1. Zunächst wird das Kreuzprodukt zu den Relationen aus dem FROM-Teil bestimmt.
2. Aus der so gebildeten Tabelle werden alle Datensätze ausgewählt, für die die WHERE-Bedingung *wahr* ergibt.
3. Die verbleibenden Datensätze werden entsprechend der GROUP-BY Klausel so gruppiert, dass Datensätze, die die gleichen Werte für die Gruppierungsattribute aufweisen, in eine Gruppe gelangen.
4. Aus den so gebildeten Gruppen werden diejenigen ausgewählt, die die HAVING-Bedingung erfüllen.
5. Für die verbleibenden Datensätze, bzw. Gruppen wird die Projektionsliste ausgewertet und die Daten werden entsprechend der Sortierreihenfolge, die im ORDER-BY angegeben ist, ausgegeben.

Es wurden bei weitem nicht alle Teile der Datenbanksprache SQL betrachtet. Eine vollständige Auseinandersetzung mit diesen Punkten würde den Seitenrahmen und den zur Verfügung stehenden Workload in dem Modul sprengen. Im Literaturverzeichnis sind einige Bücher angegeben, in denen interessierte Leser weitere Details zu SQL findet.

7 Anhang

7.1 Fallstudie FH-Info

Bei der Erläuterung der Fallstudie werden alle Aspekte, die nicht unmittelbar den Entwurf der Datenbank betreffen (wie z.B. Gestaltung der Benutzeroberfläche, Internetfähigkeit der Anwendung etc.), ausgeblendet.

Im FH-Info-System sind zu allen angebotenen Studiengängen die Bezeichnung (z.B. `Betriebswirtschaftslehrer`), die Regelstudienzeit in Semestern und der Abschluss (z.B. `B.A.`), der vergeben wird und seit wann es den Studiengang gibt, abrufbar.

Jeder Studiengang ist akkreditiert[41]. Zu jeder Akkreditierung sind das Jahr, in dem die Erst-Akkreditierung erfolgt ist, das Jahr der nächsten Re-Akkreditierung und der Name der Akkreditierungsagentur einsehbar. Jede Akkreditierung ist für einen Studiengang und jeder Studiengang muss über eine Akkreditierung verfügen! Mit jeder erfolgreichen Akkreditierung vergibt der Akkreditierungsrat eine eindeutige Nummer, unter der die Akkreditierungsurkunde bei der Agentur abrufbar ist. Diese Nummer ist zu speichern.

Für jeden eingeschriebenen Studierenden muss das System den Namen, die Anschrift, die Telefonnummer und die Matrikelnummer speichern. Die Matrikelnummern werden von der FH zentral vergeben. Es gibt keine zwei Studenten mit gleicher Matrikelnummer. Im Prinzip kann ein Studierender mehrere (verschiedene) Studiengänge studieren. Für jeden Studierenden soll gespeichert werden, seit wann sie oder er einen Studiengang studiert.

FH Info gibt Auskunft über den Aufbau von Studiengängen. Jeder Studiengang setzt sich aus mehreren Modulen zusammen. Ein Modul gehört immer zu einem Studiengang. Für jedes Modul sind die eindeutige Modulnummer, die Bezeichnung und der Workload (Credit-Points) abrufbar.

In jedem Studiengang wird jedes Semester eine Reihe von Präsenzveranstaltungen angeboten. Jede Präsenzveranstaltung ist einem Modul zugeordnet. Im FH-Info-System können für die einzelnen Präsenzveranstaltungen der Tag, der Beginn und das Ende und der Raum der Veranstaltung abgerufen werden. Jede einzelne Präsenzveranstaltung hat eine eindeutige Veranstaltungsnummer. Zu einem Modul kann es mehrere Präsenzveranstaltungen geben.

Jede Präsenzveranstaltung wird von einem Dozenten gehalten. Im FH-Info-System sind alle Präsenzdozenten mit ihrem Namen und ihrer eindeutigen E-Mail-Adresse verfügbar. Das System vergibt für jeden

41 Akkreditiert heißt, dass der Studiengang von einer unabhängigen Agentur auf die Erfüllung bestimmter Leistungs- und Qualitätsstandards hin untersucht wurde und dass die vorgegebenen Standards erfüllt sind.

neuen Dozenten eine eindeutige Nummer, unter der der Dozent im System geführt wird. Jeder Dozent kann mehrere Präsenzveranstaltungen abhalten.

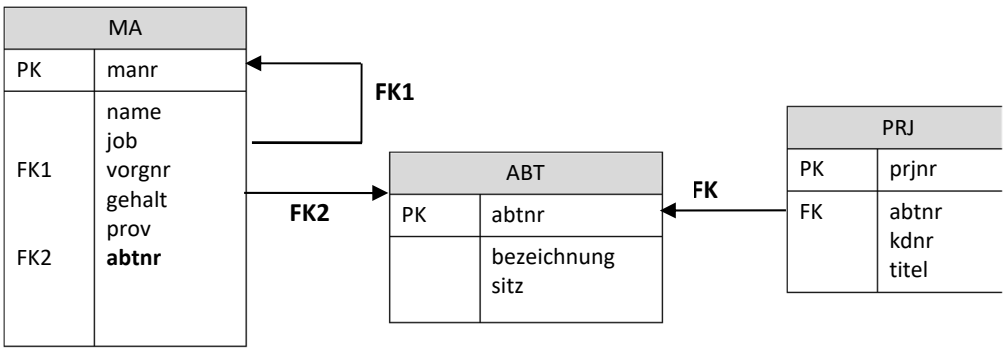
7.2 Fallstudie BestTec

Im Kapitel 4 wird die Fallstudie ***BestTec*** erstmals verwendet, um das Relationen Modell zu erläutern. Dabei werden immer nur Fragmente aus dieser Fallstudie verwendet, um einzelne Aspekte zu erläutern oder Aufgabenstellungen formulieren zu können. Die Fallstudie *BestTec* bildet die Grundlage für die Kapitel 6 und 7.

BestTec, ein weltweitführendes Unternehmen im Bereich der Automation Technology, entwickelt in Kundenprojekten maßgeschneiderte Lösung in den unterschiedlichsten Anwendungsfeldern. Darüber hinaus bietet BestTec seinen Kunden Schulungen im Einsatz und im Umgang mit den Produkten des Unternehmens und eine Vielzahl zusätzlicher Dienstleistungen rund um die Produktpalette an. Die Abteilungen von BestTec verteilen sich über verschiedene Standorte im Bundesgebiet. Die Abteilungen, die ihnen zugeordneten Mitarbeiter und Projekte sollen zukünftig in einer relationalen Datenbank gespeichert werden. Jedes Projekt wird dabei von genau einer Abteilung betreut, die die Verantwortung für das Projekt trägt.

Da im Mittelpunkt dieser Fallstudie das Relationen Modell steht, wird auf die ER-Modellierung verzichtet und nur das zugehörige Relationenschema betrachtet, das mit SQL implementiert wurde und auf der nächsten Seite abgebildet und erläutert ist.

In der so realisierten Datenbank ist bereits eine Vielzahl von Datensätzen gespeichert. In allen Ausführungen des Kapitels 6 wird davon ausgegangen, dass der auf der folgenden Seite abgebildete Datenbankzustand vorliegt.



Im Prinzip ist das Relationenschema selbsterklärend, darum hier nur einigen kurze Erläuterungen:

ABT	speichert für jede **Abteilung** die eindeutige Nummer (abtnr), die Bezeichnung (bezeichnung) und den Sitz (sitz).
PRJ	Enthält zu jedem **Projekt** die Projektnummer (prjnr), die Nummer der verantwortlichen Abteilung (abtnr), die Nummer des Kunden (kdnr) und den Projekttitel (titel).
MA	zu jedem **Mitarbeiter** wird die Personalnummer (manr), der Name (name), die Tätigkeit (job), die Personalnummer des Vorgesetzen (vorgnr), das Grundgehalt (gehalt), die Provision (prov) und die Nummer der Abteilung (abtnr), der der Mitarbeiter zugeordnet ist, gespeichert.

MA

MANr	Name	Job	VorgNr	Gehalt	Prov	AbtNr
7739	Meier	Projektleiter	7900	3200	NULL	20
7900	Karl	Manager	7656	7200	NULL	20
8801	Langer	Vertrieb	7656	4800	1200	10
7656	Hansen	Geschäftsführer	NULL	9800	NULL	10
7329	Frohn	Programmierer	7900	2900	NULL	20
8722	Werther	Programmierer	7900	2700	NULL	20
7643	Kurt	Vertrieb	8801	3100	900	10
7767	John	Controller	7656	4900	NULL	10
8800	Beier	Trainer	7783	2700	NULL	50
7621	Klaas	Trainer	7783	3200	NULL	50
7783	Kron	Manager	7656	5200	NULL	50
7768	Schulte	Vertrieb	8801	2900	500	10
8956	Dorn	Engineer	7745	3000	NULL	40
8969	Walter	Engineer	7112	2500	NULL	30
7112	Born	Manager	7656	4800	NULL	30
8322	Frei	Engineer	7745	2500	NULL	40
7769	Meyer	Engineer	7900	5200	NULL	20
8659	Sorge	Engineer	7900	5800	NULL	20
7745	Meier	Manager	7656	5500	NULL	40
8247	Holle	Engineer	7112	2900	NULL	30

ABTEILUNG

AbtNr	Bezeichnung	Sitz
10	Zentrale	Gütersloh
20	Engineering	Verl
30	Automotive	Paderborn
40	Robotik	Bielefeld
50	Education	Münster

PRJ

PRJNr	AbtNr	KDNr	titel
232	20	M341	PeakControl
244	10	T56	AutoRob
212	20	A12	HydroControl
256	20	M341	AvisyTurb

Im weiteren Verlauf des Kapitels wird von folgendem Datenbankzustand ausgegangen:

7.3 Lösungen zu den Aufgaben des Kapitels 6

Grundsätzlich gibt es zu allen Aufgaben mehr als eine richtige Lösung. Teilweise sind sehr naheliegende Alternativlösungen auch direkt angegeben. Sollten Sie also andere Lösungen als eine der hier dargestellten haben, nutzen Sie bitte die Präsenzen, um die Richtigkeit Ihrer Lösungsvorschläge zu besprechen.

Lösungen zu Aufgaben aus 6.4.3.

1.
```
SELECT name, gehalt
  FROM ma
 WHERE gehalt BETWEEN 2500 AND 4000;
```

2.
```
SELECT name, gehalt
  FROM ma
 WHERE gehalt > 2500 AND gehalt < 4000;
```

3.
```
SELECT name, gehalt, abtnr
  FROM ma
 WHERE abtnr = 10 or abtnr = 20;
```

   Alternativ:

```
SELECT name, gehalt, abtnr
  FROM ma
 WHERE abtnr IN (10, 20);
```

4.
```
SELECT name, abtnr
  FROM ma
 WHERE name like '_o__';
```

5.
```
SELECT name, gehalt, abtnr
  FROM ma
 WHERE (abtnr = 10 or abtnr = 20)
   AND gehalt BETWEEN 2500 AND 4000;
```

   Alternativ:

```
SELECT name, gehalt, abtnr
  FROM ma
 WHERE abtnr IN (10, 20)
   AND gehalt BETWEEN 2500 AND 4000;
```

6.
```
SELECT name, abtnr
  FROM ma
 WHERE (abtnr = 20 or abtnr = 30 or abtnr = 40)
   AND name like '_o__';
```

Alternativ:

```
SELECT name, abtnr
  FROM ma
 WHERE abtnr in (20, 30, 40)
   AND name like '_o__';
```

7.
```
SELECT name, abtnr, gehalt
  FROM ma
 WHERE abtnr IN (10, 20)
   AND (job = 'Manager' or gehalt > 5000);
```

8.
```
SELECT name, abtnr, job, gehalt
  FROM ma
 WHERE gehalt < 5000
   AND job NOT IN ('Programmierer',
                   'Administrator')
    OR (job IN ('Trainer', 'Controller');
```

Alternativ:

```
SELECT name, abtnr, job, gehalt
  FROM ma
 WHERE gehalt < 5000
   AND job != 'Programmierer'
   AND job != 'Administrator'
    OR (job = 'Trainer' or job = 'Controller');
```

Achtung: Eine Klammerung der ersten mit AND verknüpften Teilbedingungen ist nicht erforderlich, da AND stärker bindet als OR. Sollte die Bedingungen dennoch geklammert sein, ist dies ebenfalls korrekt!

Lösungen zu Aufgaben aus 6.5.1.

9.
```
SELECT name, gehalt, abtnr
  FROM ma
 WHERE abtnr IN (10, 20)
   AND gehalt BETWEEN 2500 AND 4000
ORDER BY abtnr ASC, gehalt DESC;
```

10.
```
SELECT name, abtnr, job, gehalt
  FROM ma
 WHERE gehalt < 5000
   AND job NOT IN ('Programmierer',
                   'Administrator')

    OR (job IN ('Trainer', 'Controller')
ORDER BY job ASC, abtnr ASC, gehalt ASC;
```

Lösungen zu Aufgaben aus 6.6.1.

Bei den folgenden Lösungen stellen wir den Spaltennamen nur dann einen Tabellennamen voran, wo dies zwingend erforderlich ist. Alternativ kann natürlich jedem Spaltennamen der zugehörige Tabellenname vorangestellt werden.

11.
```
SELECT abt.abtnr, bezeichnung, manr, name
  FROM ma, abt
 WHERE ma.abtnr = abt.abtnr
   AND sitz = 'Münster';
```

12.
```
SELECT abt.abtnr, bezeichnung, manr, name
  FROM ma, abt
 WHERE ma.abtnr = abt.abtnr
   AND (bezeichnung = 'Education' or
        bezeichnung = 'Zentrale');
```

13.
```
SELECT abt.abtnr, bezeichnung, name, gehalt
  FROM ma, abt
 WHERE ma.abtnr = abt.abtnr
   AND (bezeichnung = 'Education' or
        bezeichnung = 'Zentrale')
       AND gehalt > 3000
ORDER BY bezeichnung ASC, gehalt DESC;
```

Lösungen zu Aufgaben aus 6.7.2.

14.
```
SELECT COUNT(*) AS ANZAHL
  FROM ma;
```

15.
```
SELECT COUNT(*) AS ANZAHL
  FROM ma;
WHERE abtnr in (30, 40);
```

16.
```
SELECT COUNT(*) AS ANZAHL
  FROM ma, abt;
WHERE ma.abtnr = abt.abtnr
   AND (sitz = 'Gütersloh' OR
        sitz = 'Bielefeld');
```

17.
```
SELECT COUNT(*)    AS ANZAHL,
       AVG(gehalt) AS D_GEHALT
       SUM(gehalt) AS SUMME_GEHALT
  FROM ma, abt;
WHERE ma.abtnr = abt.abtnr
   AND sitz IN ('Gütersloh', 'Bielefeld');
```

Lösungen zu Aufgaben aus 6.7.4.

18.
```
SELECT abtnr, COUNT(*) AS ANZAHL
  FROM ma
GROUP BY abtnr;
```

19.
```
SELECT bezeichnung, COUNT(*) AS ANZAHL
  FROM ma, abt
 WHERE ma.abtnr = abt.abtnr
GROUP BY bezeichnung;
```

20.
```
SELECT abt.abtnr, bezeichnung,
       AVG(gehalt) AS D_GEHALT
  FROM ma, abt
 WHERE ma.abtnr = abt.abtnr
   AND sitz in ('Gütersloh', 'Bielefeld',
                'Paderborn')
GROUP BY abt.abtnr, bezeichnung;
```

21.
```
SELECT job, COUNT(*) AS ANZAHL,
       AVG(gehalt) AS D_GEHALT
  FROM ma
GROUP BY job
ORDER BY AVG(gehalt);
```

Lösungen zu Aufgaben aus 6.7.6.

22.
```
SELECT bezeichnung, COUNT(*) AS ANZAHL
  FROM ma, abt
 WHERE ma.abtnr = abt.abtnr
GROUP BY bezeichnung
HAVING COUNT(*) < 5;
```

23.
```
SELECT job, COUNT(*) AS ANZAHL,
       AVG(gehalt) AS D_GEHALT
  FROM ma
GROUP BY job
HAVING COUNT(*) > 4
    OR AVG(gehalt) > 4000;
```

Lösungen zu Aufgaben aus 6.8.1.

24.

a.
```
SELECT kdnr, titel
        FROM prj, abt
       WHERE prj.abtnr = abt.abtnr
         AND sitz = 'Gütersloh';
```

b.
```
   SELECT kdnr, titel
        FROM prj
       WHERE abtnr IN (SELECT abtnr
                         FROM abt
                        WHERE sitz = 'Güters-
   loh');
```

25.
```
SELECT abtnr, bezeichnung
  FROM abt
 WHERE abtnr NOT IN (SELECT abtnr
                       FROM ma
                      WHERE job = 'Engineer');
```

26.
```
SELECT manr, name, gehalt
  FROM ma
 WHERE gehalt > (SELECT AVG(gehalt)
                   FROM ma);
```

27.
```
SELECT bezeichnung, count(*) AS ANZAHL
  FROM ma
GROUP BY bezeichnung
HAVING count(*) > (SELECT COUNT(*)
                     FROM ma
                    WHERE abtnr = 10);
```

28.
```
SELECT bezeichnung, AVG(gehalt) AS D_GEHALT
  FROM ma
GROUP BY bezeichnung
HAVING count(*) > (SELECT COUNT(*)
                     FROM ma, abt
                       WHERE ma.abtnr = abt.abtnr
                         AND sitz = 'Münster');
```

Alternativ:

```
SELECT bezeichnung, AVG(gehalt) AS D_GEHALT
  FROM ma
GROUP BY bezeichnung
HAVING count(*) > (SELECT COUNT(*)
                     FROM ma
                   WHERE abtnr IN (SELECT abtnr
                                     FROM abt
                                    WHERE sitz =
                                    'Münster'));
```

7.4 Syntax der SQL-Anweisungen

Die folgenden Regeln fassen die Syntax der eingeführten SQL-Anweisungen zusammen.

Select-Anfrage:

```
SELECT [DISTINCT] Projektionsliste
  FROM FROM-Klausel
 WHERE Bedingung
 GROUP BY Gruppenbildung
 HAVING Bedingung
 ORDER BY Sortierung
```

Projektionsliste:

1. *
2. $Werteausdruck_1$ [**AS** $Überschrift_1$]
   ,...,
   $Werteausdruck_n$ [**AS** $Überschrift_n$]

FROM-Klausel:

$Tabellenname_1$,..., $Tabellenname_n$

Werteausdruck:

- *[Tabellenname.]Spaltenname*
- *Wert*

  Werte werden entsprechend Syntax des jeweiligen Wertebereichs geschrieben.
- *Berechnung*

  Berechnungen werden mittels Operationen, die für die einzelnen Wertebereich definiert sind, gebildet (siehe Kapitel 2).
- **COUNT(*)**
- *Gruppenfunktion*([**DISTINCT**] *Werteausdruck*)

  Mögliche Gruppenfunktionen sind:

  AVG, SUM, COUNT, MIN, MAX

Bedingung:

Einfache Bedingungen, die den direkten Wertevergleich erlauben:

- *Werteausdruck vOp Werteausdruck*

  Mögliche Vergleichsoperatoren an Stelle von vOp *sind:*
  =, <=, >=, <, >, <>

- *Werteausdruck* **[NOT] IN (***$Wert_1$,..., $Wert_n$***)**
- *Werteausdruck* **BETWEEN** *$Wert_1$* **AND** *$Wert_2$*
- *Werteausdruck* **LIKE** *Zeichenkettenmuster*

Bedingungen mit Unteranfragen:

- *Werteausdruck* **[NOT] IN (***Select-Anfrage***)**
- *Werteausdruck vOp* **(***Select-Anfrage***)**

Neben diesen Bedingungen können Klammern gesetzt werden und AND, OR und NOT verwendet werden:

- **(***Bedingung***)**
- *Bedingung* **AND** *Bedingung*
- *Bedingung* **OR** *Bedingung*
- **NOT** *Bedingung*

Einschränkung für den WHERE-Teil:

In der Bedingung im WHERE-Teil dürfen nur Werteausdrücke verwendet werden, die keine Gruppenfunktionen enthalten!

Gruppenbildung:

$[Tabellenname_1.]Spaltenname_1$,

...,

$[Tabellenname_n.]$ $Spaltenname_n$

Sortierung:

$Werteausdruck_1$,[Reihenfolge]

,...,

$Werteausdruck_n$ [Reihenfolge]

Als Reihenfolge stehen zur Verfügung:

ASC *(aufsteigend)*
DESC *(absteigend).*

Literaturempfehlungen

Die in diesem kommentierten Literaturverzeichnis aufgeführten Empfehlungen erlauben eine weitergehende und vertiefende Auseinandersetzung mit den Themen ER-Modellierung, Relationen Modell, Schema Transformation und SQL. Diese Themen sind Gegenstand jedes klassischen Datenbanklehrbuchs und nehmen dort viele Seiten ein.

R. Adams: SQL – Der Grundkurs für Ausbildung und Praxis, Hanser 2019
Der Autor führt in das Thema Datenbanken sehr praxisorientiert ein. Das Vorgehen unterscheidet sich dabei allerdings an vielen Stellen von dem in dieser Lerneinheit gewählten Weg.

A. Heuer, G. Saake, K.-U. Sattler: Datenbanken – Konzepte und Sprachen, MITP 2018
Da der Datenbankentwurf einen Schwerpunkt des Buches darstellt, werden ER-Modell, Relationen Modell, Schematransformation und Normalisierung sehr ausführlich und anschaulich erläutert. Die Einführung von SQL erfolgt auf Basis eines umfangreichen Beispiels.

A. Kemper, A. Eickler: Datenbanksysteme – Eine Einführung, Oldenbourg 2015
Der Bereich der Schematransformation wird nicht sehr ausführlich behandelt. Insbesondere wird nicht auf die Einhaltung der Informationskapazität eingegangen. Der konzeptuelle und der relationale Datenbankentwurf werden dafür ausführlicher behandelt und geht die Inhalte dieser Lerneinheit hinaus. Die Vorstellung von SQL nimmt keinen sehr breiten Raum ein.

W. Panny, A. Taudes: Einführung in den Sprachkern von SQL-99, Springer-Verlag 2000
Im Mittelpunkt des Buches steht die Sprache SQL. Auch wenn das Buch bereits 2009 erschienen ist, findet sich hier eine umfangreiche und sehr detaillierte Erläuterung der gängigen SQL-Anweisungen. Dieses Buch ist in vielen Hochschulen als eBook über springerlink.com verfügbar.

P. Sauer: Informationsmodellierung, erschienen als Kapitel 3 und 4 im Taschenbuch Datenbanken (Hrsg. T. Kudraß), Hanser 2015
Auf relativ wenigen Seiten wird das ER-Modell, das Relationen Modell und die Schematransformation überblicksartig vorgestellt. Dieses Buch ist mehr als Nachschlagewerk und weniger als Lehrbuch geeignet.

E. Schicker: Datenbanken und SQL – Eine praxisorientierte Einfürhung, Springer/Vieweg 2017
Dieses Buch wählt einen sehr pragmatischen Zugang zu dem Thema Datenbanken und verzichtet auf unnötige theoretischen und mathematische Ausführungen. Dieses Buch ist in vielen Hochschulen als e-Book über springerlink.com verfügbar.