Datenbanken und Tabellen aus dem Entity-Relationship-Modell, Structured Query Language (SQL))

Bernd Blümel, Volker Klingspor, Christian Metzger

Version: 13. März 2012

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

1.1 Speicherung von Daten in Datenbanken

Im Computerzeitalter werden Mengen von Daten gespeichert, z.B. die Verbindungsdaten Ihres Internetzugangs sowie die Ihrer Handytelefonate. Auch an der Hochschule Bochum werden Daten gespeichert, z.B. die Prüfungsleistungen der Studierenden sowie deren Adressen. Die Speicherung dieser Daten kann in Textdateien, Exceltabellen oder in Datenbanken erfolgen. Eine Exceltabelle sowie eine Tabelle einer Datenbank können ganz ähnlich, manchmal sogar identisch aussehen. Warum Excel aber dennoch nicht für Datenspeicherung geeignet ist, werden wir Ihnen nachfolgend erklären.

Angenommen, wir wären Mobilfunkprovider und hätten 30 Millionen Kundinnen. Jede Kundin telefoniert pro Tag zwölf Mal. Für die Abrechnung müssen wir den Beginn sowie das Ende des Telefonats und die Gesprächszone speichern. Das sind pro Tag: 30.000.000 x 12 = 360.000.000 Millionen Datensätze. Sind solche Datenmengen in Excel speicherbar? Probieren Sie es doch einfach aus ;-).
Die maximale Zeilenanzahl für eine Exceltabelle wird nicht ausreichen um 360.000.000 Datensätze abzuspeichern. Aber selbst wenn dies möglich wäre, also wenn man Milliarden von Zeilen speichern könnte, stoßen wir relativ schnell an die nächste Grenze von Excel, die Netzwerkfähigkeit. Wenn Sie eine Exceltabelle benutzen, können Sie einer weiteren Person Zugriff darauf geben? Grundsätzlich ja, nämlich indem Sie diese Exceltabelle der betreffenden Person z.B. per E-Mail oder auf USB-Stick schicken, oder indem Sie über Collaboration-Tools[1] zusammen an der Exceltabelle arbeiten. Wenn Sie die Exceltabelle versenden, können Sie in der Zwischenzeit nicht an der Liste arbeiten, da Sie sonst unterschiedliche Versionen haben. Würden Sie zwischenzeitlich einen Datensatz nachtragen, wäre dieser nicht automatisch in der versendeten Exceltabelle vorhanden. Man könnte nun stundenlang weitere Probleme aufzeigen, welche bei der Speicherung von Daten mit Excel entstehen, das Ergebnis wäre immer das gleiche: Excel eignet sich für Datenjonglage[2], aber nicht zur professionellen Datenspeicherung. Dies wird schon durch die Bezeichnung seitens des Herstellers ausgedrückt. Excel wird als Tabellenkalkulation verkauft. Es dient somit zum Berechnen von Daten und nicht zu deren Speicherung. Wobei, rechnen können Sie mit den in Datenbanken abgespeicherten Daten auch. Es ist sogar möglich mit Excel auf eine Datenbank zu konnektieren und sich Daten für Berechnungen direkt aus der Datenbank zu holen.

Professionelle Datenspeicherung ist nur in Datenbanken möglich. Bei dem Wort Datenbanken fällt den meisten (warum auch immer) direkt Microsoft Access ein. Microsoft Access ist aber keine Datenbank, sondern nur die Software um eine Datenbank zu erstellen (ein sogenanntes Datenbankmanagementsystem). Teilweise kann man bei Access Datenbanken per Mausklick erstellen. Das funktioniert aber nur dann, wenn die Vorlage genau das abbildet was Sie haben möchten, z.B. eine CD-Verwaltung. Möchten Sie aber in Access z.B. Daten von Bilanzen abspeichern, so stellen Sie fest dass es hierfür keine Vorlage gibt (und schon gehen die Probleme los ;-)). Aber keine Panik, wir zeigen Ihnen wie Sie vorgehen müssen um auch dieses Problem zu lösen.

Sie werden hier lernen, wie man relationale Datenbanken entwirft und wie Sie dann, basierend auf einem Modell, die benötigten Tabellen ableiten.

[1] Netzwerkfähige Software zur Teamarbeit
[2] Schönes Wort, nicht?

1.2 Einleitendes Beispiel

Fangen wir direkt mit einem Beispiel für eine schlechte Struktur an:

KundeNr	Name	Plz	Stadt	AuftragDatum	AuftragBeschreibung
1	Schulz GmbH	44701	Bochum	01.09.2014	Werkhalle streichen
2	Meier AG	47543	Bochum	03.09.2014	Teppichboden verlegen
1	Schulz GmbH	44701	Bochum	05.09.2014	Laminat verlegen
3	Fischer	44787	Bochum	08.09.2014	Tapezieren
1	Schulz GmbH	44702	Bochum	09.09.2014	Werkhalle Boden verlegen
2	Meier GmbH	47543	Bochum	10.09.2014	Laminat verlegen

Abbildung 1.1
Beispiel für eine schlechte Datenstruktur

Sie sehen hier eine Exceltabelle. Betrachten Sie diese, fallen Ihnen möglicherweise direkt folgende Mängel auf:

1.3 Mängel in der Datenstruktur

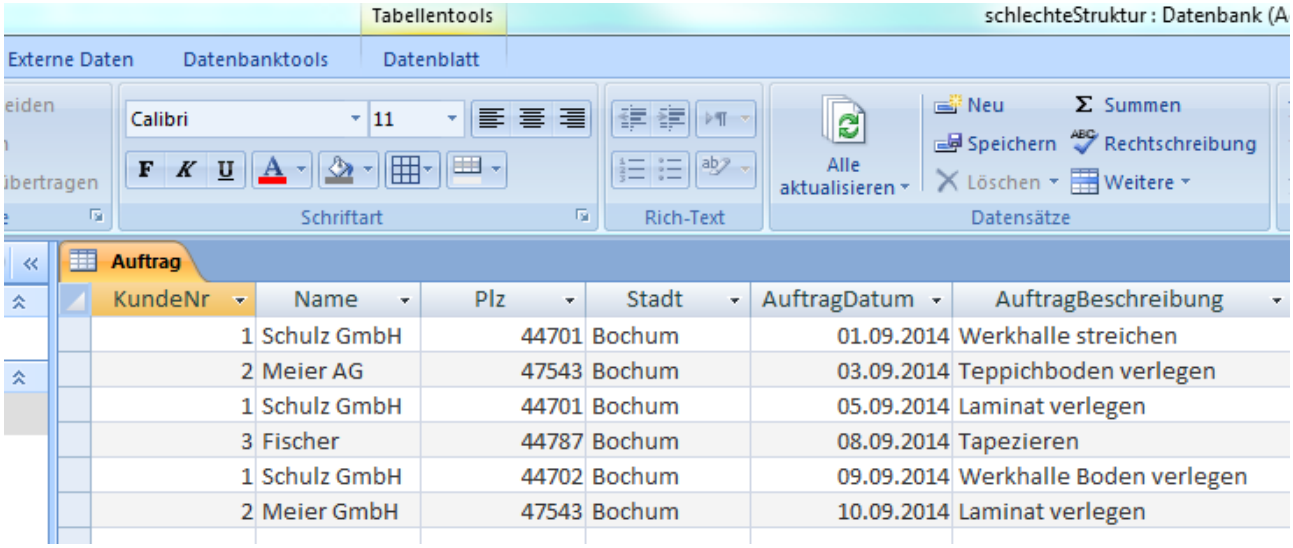
- Datenredundanz
  dieselben Daten werden mehrfach gespeichert (Bsp.: die Schulz GmbH)
- Inkonsistenz
  für dasselbe Objekt in der Realität liegen unterschiedliche (sich widersprechende) Daten in der Datenbank vor (Bsp.: PLZ der Schulz GmbH, Meier AG sowie Meier GmbH)

Auch treten Anomalien auf:

- Löschanomalie
  Kunde wird beim Löschen seines letzten Auftrags mit gelöscht
  (Bsp.: Wenn Auftrag mit der AuftragBeschreibung [3] “Tapezieren“ gelöscht wird, gibt es keine Informationen zum Kunden Fischer mehr)
- Einfügeanomalie
  Ein Kunde kann nur direkt mit einem Auftrag angelegt werden
- Änderungsanomalie
  Änderung eines Kunden in mehreren Datensätzen nötig (Beispiel: Ändert sich die Postleitzahl der Schulz GmbH müssen drei Datensätze manuell geändert werden)

Die Praxis sieht manchmal so aus, dass irgendwann einmal eine Exceltabelle existierte und diese dann, durch einen Mitarbeiter welcher Microsoft Access entdeckt hat, in eine Datenbank ”umgewandelt” wurde. Meistens wird hierfür der Importassistent von Microsoft Access benutzt. Unser Exceltabelle könnte man auch problemlos in Microsoft Access importieren. Das tun wir jetzt einfach mal. Mittels dem Importassistenten geht das sehr flott und schon ist aus unserer Exceltabelle eine Tabelle in einer Datenbank geworden (Abb.1.2).
Erkennen Sie einen Unterschied zur Exceltabelle in Abbildung 1.1? Nein? Gut, es gibt auch keinen. Nur weil diese Exceltabelle von Microsoft Access importiert wurde, entsteht noch keine Datenstruktur und auch keine strukturierte Datenbank. Wie kann man nun diese Exceltabelle so verändern, dass eine gute Datenstruktur entsteht? Betrachtet man die Liste, so fällt auf, dass hier zwei Objekte abgespeichert werden, einmal die Kunden und einmal deren Aufträge. Das ist der Fehler bei dieser Liste, zwei verschiedene Objekte werden zusammen abgespeichert. Wie behebt man nun diesen Fehler? Man trennt die beiden Objekte.

[3] damit ist die Spalte in der Tabelle gemeint



KundeNr	Name	Plz	Stadt	AuftragDatum	AuftragBeschreibung
1	Schulz GmbH	44701	Bochum	01.09.2014	Werkhalle streichen
2	Meier AG	47543	Bochum	03.09.2014	Teppichboden verlegen
1	Schulz GmbH	44701	Bochum	05.09.2014	Laminat verlegen
3	Fischer	44787	Bochum	08.09.2014	Tapezieren
1	Schulz GmbH	44702	Bochum	09.09.2014	Werkhalle Boden verlegen
2	Meier GmbH	47543	Bochum	10.09.2014	Laminat verlegen

Abbildung 1.2
Import einer Exceltabelle in Microsoft Access per Assistent

1.4 Mängelbeseitigung

Zieht man gedanklich einen Strich zwischen den Spalten Stadt und fügt eine Spalte AuftragNr ein, so hat man die Exceltabelle und damit auch die beiden Objekte sinnvoll aufgeteilt. Der Teil links enthält die Kundendaten und der Teil rechts die Auftragsdaten (Abb. 1.4).

KundeNr	Name	Plz	Stadt	AuftragDatum	AuftragBeschreibung
1	Schulz GmbH	44701	Bochum	01.09.2014	Werkhalle streichen
2	Meier AG	47543	Bochum	03.09.2014	Teppichboden verlegen
1	Schulz GmbH	44701	Bochum	05.09.2014	Laminat verlegen
3	Fischer	44787	Bochum	08.09.2014	Tapezieren
1	Schulz GmbH	44702	Bochum	09.09.2014	Werkhalle Boden verlegen
2	Meier GmbH	47543	Bochum	10.09.2014	Laminat verlegen

Abbildung 1.3
Aufteilung der verschiedenen Objekte

KundeNr	Name	Plz	Stadt
1	Schulz GmbH	44701	Bochum
2	Meier AG	47543	Bochum
1	Schulz GmbH	44701	Bochum
3	Fischer	44787	Bochum
1	Schulz GmbH	44702	Bochum
2	Meier GmbH	47543	Bochum

AuftragDatum	AuftragBeschreibung
01.09.2014	Werkhalle streichen
03.09.2014	Teppichboden verlegen
05.09.2014	Laminat verlegen
08.09.2014	Tapezieren
09.09.2014	Werkhalle Boden verlegen
10.09.2014	Laminat verlegen

Abbildung 1.4
Tabelle Kunde und Tabelle Auftrag

KundeNr	Name	Plz	Stadt
1	Schulz GmbH	44701	Bochum
2	Meier AG	47543	Bochum
3	Fischer	44787	Bochum

Abbildung 1.5
Bereinigte Tabelle Kunde

AuftragNr	AuftragDatum	AuftragBeschreibung
1	01.09.2014	Werkhalle streichen
2	03.09.2014	Teppichboden verlegen
3	05.09.2014	Laminat verlegen
4	08.09.2014	Tapezieren
5	09.09.2014	Werkhalle Boden verlegen
6	10.09.2014	Laminat verlegen

Abbildung 1.6
Tabelle Auftrag

Nachdem wir die Objekte jeweils in einer eigenen Tabelle gespeichert haben, betrachten wir die linke Tabelle von Abbildung 1.4, nennen wir sie ”Kunde”. Die doppelte Speicherung der Datensätze ist immer noch vorhanden, aber die doppelten Datensätze lassen sich jetzt einfach löschen, das Ergebnis ist dann die Tabelle “Kunde“ in Abbildung 1.5.
Die Tabelle ”Auftrag” auf der rechten Seite (Abb. 1.6) hat keine doppelten Datensätze, wir müssen in der Tabelle keine Datensätze löschen. Als Ergebnis haben wir jetzt zwei Tabellen, Kunde sowie Auftrag. Schauen wir auf die Tabelle ”Auftrag”, so stellen wir fest, dass wir jetzt nicht wissen welcher Auftrag von welchem Kunden kam. Das liegt daran, dass der Kunde nun nicht mehr neben dem Auftrag steht. Das ist ein richtiges Problem, wenn man nicht weiß, welcher Auftrag von welchem Kunden erteilt wurde. Wie können wir dieses Problem lösen? Ganz einfach, wir fügen der Tabelle Auftrag eine Spalte hinzu und schreiben in diese die Kundennummer.
Nun können wir anhand der Kundennummer eine Verbindung zu der Tabelle Kunde herstellen und wissen welcher Kunde

AuftragNr	AuftragDatum	AuftragBeschreibung	KundeNr
1	01.09.2014	Werkhalle streichen	1
2	03.09.2014	Teppichboden verlegen	2
3	05.09.2014	Laminat verlegen	1
4	08.09.2014	Tapezieren	3
5	09.09.2014	Werkhalle Boden verlegen	1
6	10.09.2014	Laminat verlegen	2

Abbildung 1.7
Fremdschlüsselspalte in Tabelle Auftrag

den Auftrag erteilt hat.

Fachlich korrekt würde man sagen:
Der Primärschlüssel aus der Kundentabelle wird Fremdschlüssel in der Auftragstabelle.
Aber was ist nun ein Primärschlüssel?
Ein Primärschlüssel identifiziert einen Datensatz eindeutig. Als Primärschlüssel wird entweder eine “natürliche“ Nummer, wie z.B. Ihre Matrikelnummer oder eine aufsteigende “künstliche“ Nummer benutzt.
Schauen Sie sich Ihren Personalausweis an: Die Nummer des Personalausweises ist auch ein Primärschlüssel, da diese

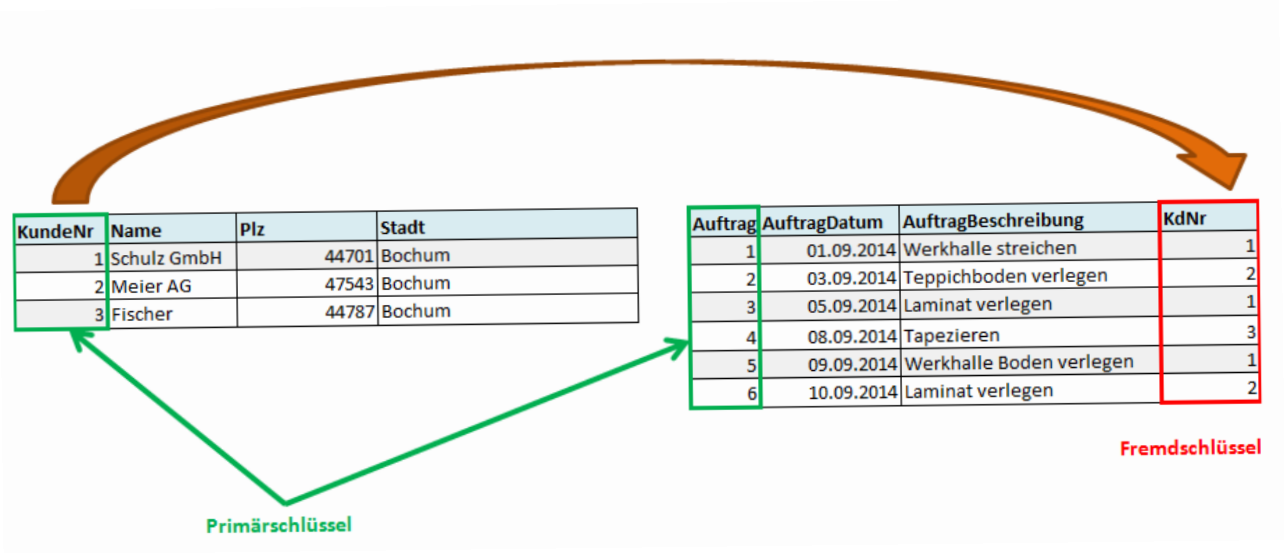


Abbildung 1.8
Transfer Primärschlüssel Fremdschlüssel

Personalausweisnummer nur einmal in Deutschland vergeben wurde[4]. Ihre Handynummer ist auch einmalig. Somit wäre Ihre Handynummer auch ein geeigneter Primärschlüssel falls man Daten über Sie speichern möchte. Der Primärschlüssel aus der Kundentabelle wird zum Fremdschlüssel der Auftragstabelle, aber könnte man nicht auch die Auftragsnummer in der Kundentabelle abspeichern? Warum muss die Kundennummer in die Auftragstabelle? Wir zeigen Ihnen in der Abbildung 1.9 warum es keinen Sinn ergibt den Primärschlüssel aus der Auftragstabelle, die Auftragsnummer, als Fremdschlüssel in die Kundentabelle einzufügen: Für jeden Auftrag eines Kunden ist eine zusätzliche Spalte anzulegen, d.h. bei

KundeNr	Name	Plz	Stadt	Auftrag Nr	AuftragNr	AuftragNr
1	Schulz GmbH	44701	Bochum	1	3	5
2	Meier AG	47543	Bochum		2	6
3	Fischer	44787	Bochum	4		

Abbildung 1.9
Beispiel für falschen Fremdschlüsseltransfer

jedem Auftrag wäre die Struktur der Tabelle zu ändern. Das macht keinen Sinn und deswegen wird die Kundennummer als Fremdschlüssel in die Auftragstabelle eingefügt.

Sie sahen nun die Modellierung einer einfachen Auftragsverwaltungsdatenbank und kennen jetzt die Begriffe Primär- sowie Fremdschlüssel. Eine Tabelle hat aber noch weitere Bestandteile, welche nachfolgend grafisch in Abbildung 1.10 erläutert werden. Die Sortierung des Primärschlüssels bedeutet übrigens nicht, dass der Datensatz an dieser Position in der Tabelle steht. Möchte man sich nun alle Aufträge des Kunden mit der Kundennummer 1 anzeigen lassen, so definiert man eine Abfrage die umgangssprachlich lautet:
”Zeige mir alle Aufträge der Tabelle Auftrag bei denen in der Spalte KundeNr eine 1 steht.”
Möchte man alle Aufträge anzeigen bei denen Laminat verarbeitet wurde definiert man eine Abfrage:
”Zeige mir alle Datensätze der Tabelle Auftrag bei denen in der Spalte Auftragsbeschreibung der Text ”Laminat” steht.”
Die korrekte Syntax der Abfragen zeigen wir Ihnen im Kapitel “SQL“ weiter hinten im Script.

[4]Wir gehen davon aus, dass Ausweisnummern in anderen Ländern auch nur einmal vergeben werden

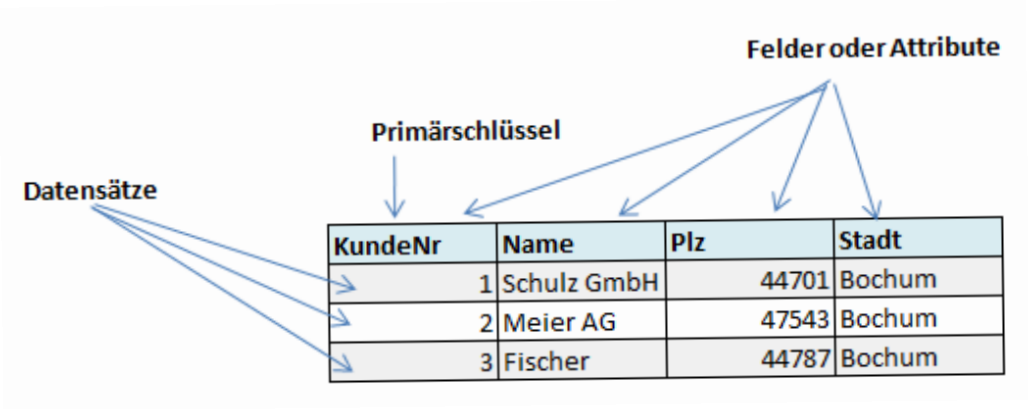


Abbildung 1.10
Aufbau einer Tabelle

1.5 Zusammenfassung

Bei der Speicherung von Daten ist es wichtig die verschiedenen Objekte in verschiedenen Tabellen zu speichern. Tabellen bestehen aus Zeilen sowie Spalten. In den Zeilen stehen die Datensätze, in den Spalten die jeweiligen Einzelwerte. Jeder Datensatz hat eine eindeutige ID, den Primärschlüssel. Mittels dieses Primärschlüssels ist der Datensatz eindeutig identifizierbar. Der Primärschlüssel muss aber nicht zwingend eine Zahl sein, es empfiehlt sich aber ein numerischer Wert. Tabellen können zueinander in Beziehung stehen. Ist dies der Fall so werden die Tabellen miteinander verknüpft. Bei einer einfachen Verknüpfung ”wandert” der Primärschlüssel der einen Tabelle als Fremdschlüssel in die andere Tabelle. Die Richtung in welche der Primärschlüssel wandert ist entscheidend. Im Kapitel ”Tabellen ableiten” erfahren Sie dann mehr zu dem Thema.

Kapitel 2

ER-Modellierung

2.1 Datenmodelle

Wir zeigen Ihnen nun die Vorgehensweise um ein Abbild einer Szene der realen Welt, wie z.B. eine Auftragserteilung, in einer Datenbank strukturiert abzuspeichern. Um dieses Vorhaben erfolgreich umzusetzen benötigten wir zuerst ein Modell. Nein, kein menschliches, sondern eins aus der Informatik :-). In der (Wirtschafts-) Informatik werden Modelle benutzt um Sachverhalte grafisch darzustellen. Da Modelle immer Daten, sowie die Beziehungen der Daten zueinander enthalten, nennt man sie Datenmodelle. Datenmodelle sind eine abstrakte Sicht auf eine Szenerie der realen Welt, bei der nur die für die Szenerie relevanten Informationen dargestellt werden. Datenmodelle können durch natürliche Sprache, mathematische Formeln oder auch grafisch beschrieben werden. Für die Modellierung von Datenbanken wird ein grafisches Modell benutzt. Warum werden nun Datenmodelle grafisch dargestellt? Weil grafische Datenmodelle:

- eine präzisere Aussagekraft als Text haben. Es wäre manchmal schwierig bestimmte Konstellationen mit Text eindeutig zu beschreiben,
- helfen, bei umfangreichen Projekten die Übersicht zu behalten
- einfach zu erfassen sind, auch von fachunkundigen Personen z.B. Managern ;-),
- einfach zu ändern sind im Projektverlauf, falls dies aufgrund geänderter Anforderungen oder falschem Verstehens notwendig wird,
- sich hervorragend für Dokumentation (und man sollte immer dokumentieren) eignen,
- als Diskussiongrundlage verwendet werden können und
- dem Informationsaustausch dienen

Um diese ganzen Vorteile nutzen zu können, ist es wichtig, sich bei Modellen an gewisse Formalismen zu halten. Für jedes Modell gibt es bestimmte grafische Elemente, welche verwendet werden müssen. Verwenden Sie andere Elemente als die vorgesehenen, kann ihr Modell nicht richtig interpretiert werden. Es gibt verschiedene Arten grafischer Datenmodelle. Bei der ER-Modellierung gibt es z.B. die Chen-, Krähenfuß sowie die UML-Notation. Alle Notationen zeigen dieselben Informationen, nur die grafische Darstellung ist unterschiedlich. Beherrschen Sie eine Notation, können Sie jede andere Notation schnell erlernen. Wir verwenden hier in diesem Skript die UML-Notation[1] weil diese international standardisiert sowie zukunftssicher ist.
Haben Sie das Datenmodell entwickelt, ist es völlig irrelevant in welchem Datenbankmanagementsystem Sie die Datenbank anlegen. Sie werden nun überlegen was wohl ein Datenbankmanagementsystem ist? Das ist die Software welche Sie zum Anlegen und Verwalten der Datenbank sowie der Tabellen benutzen. Gemeint ist z.B. MySQL, Oracle, MS SQL Server, IBM DB2 usw.. Es ist völlig egal ob Sie die Datenbank in MySQL oder Oracle anlegen, das Ausgangsdatenmodell ist immer das Gleiche. Das bedeutet: Haben Sie z.B. ein ER-Modell für eine Kundenverwaltung entwickelt, können Sie die Kundenveraltungsdatenbank in MySQL anlegen und mit dem gleichen ER-Modell können Sie die Kundenverwaltungsdatenbank auch in Oracle implementieren.

[1] UML ist die Abkürzung für Unfied-Modelling-Language

Achso, fast hätten wir es vergessen: Der Hauptgrund warum wir das Datenmodell benötigen bei der Datenbankentwicklung ist, dass wir damit anschließend die zur strukturierten Speicherung der Daten erforderlichen Tabellen ableiten können.

2.2 Bestandteile eines Entity-Relationship-Modells

Bestandteil	**Erklärung**
Attribut	Eigenschaften einer Entität, z.B. Größe, Gewicht, Stromverbrauch, Auftragsdatum
Beziehung	Relation, d.h. reale Beziehung zwischen zwei Entitäten, z.B. Kunde erteilt Auftrag, Prof. hält Vorlesung, Student schreibt Klausur
Entität	eindeutig identifizierbares Objekt,z.B. Herr Müller, der Kölner Dom, die Banane
Entitätstyp	Oberbegriff für eine Menge von Entitäten, z.B. Kunde, Bauwerk, Obst
Kardinalität	Verhältnis von Entität A (z.B. Kunde) zu Entität B (z.B. Auftrag) oder anders ausgedrückt: Wieviele Entitäten sind an einer Beziehung beteiligt? • Anzahl der möglicherweise erteilbaren Aufträge durch einen Kunden • Anzahl der möglicherweise besuchbaren Vorlesung durch einen Studenten • Anzahl der Studierenden welche in einen Hörsaal passen

Tabelle 2.1
Bestandteile eines Entity-Relationship-Modells

2.3 Erstes Datenmodell

Basierend auf dem folgenden Beispiel werden wir das erste Datenmodell entwickeln: Sie erteilen den Auftrag Ihre Wohnung neu zu streichen. Etwaiges Material (Farbe, Tapeten, Pinsel usw.) soll vom Auftragnehmer bereitgestellt werden. Diesen Sachverhalt drücken wir ein bisschen abstrakter aus:
Ein Kunde kann Aufträge erteilen. Ein Auftrag kann Artikel enthalten.

Abbildung 2.1
Beteiligte an einem Auftrag

Wir stellen uns demnach vor:
Familie Mustermann erteilt den Auftrag ihr Haus zu streichen. Für die Ausführung des Auftrags werden mehrere Eimer Farbe sowie verschiedene Pinsel benötigt.
Wir müssen nun speichern:
Familie Mustermann ist der Auftraggeber, der Auftrag besteht darin ein Haus zu streichen und für die Leistungserbringung werden ein paar Eimer Farbe sowie Pinsel benötigt. Sie wundern sich vielleicht warum kein Handwerker in dem Beispiel aufgeführt ist, welcher den Auftrag ausführt. Das hat folgende Gründe:

- In dem Aufgabentext ist er nicht explizit aufgeführt
- Es muss nicht unbedingt ein Handwerker sein, sondern könnte auch ein beliebiger Dienstleister sein. Das Beispiel mit ”Haus streichen” ist willkürlich gewählt, wir könnten auch ein Beispiel ”Frau X erteilt den Auftrag eine Statue zu meißeln” benutzen.
- Stellen Sie sich vor Sie sind der Auftragnehmer, dann erhalten Sie von Familie Mustermann den Auftrag das Haus zu streichen. Damit Sie eine Übersicht über die erhaltenen Aufträge, sowie über das hierfür benötigte Material haben, speichern Sie die Auftrags- und die Kundendaten ab.

Nun kommt ein weiterer Auftrag: Herr Murx erteilt den Auftrag in seinem Garten einen Baum zu fällen. An der Stelle an der der Baum stand, soll ein Gartenteich entstehen. Für diesen wird Teichfolie benötigt. Diese Daten müssen wir jetzt irgendwie abspeichern. Wie eine Tabelle aussieht wissen Sie bereits. Wenn wir in einer Tabelle alles (Auftraggeber, Auftrag sowie Artikel) abspeichern, erhalten wir diverse Anomalien. Somit brauchen wir eine andere Vorgehensweise. Am besten wir malen diese beiden Sachverhalte einmal auf, jedes Objekt malen wir in ein Rechteck. Das Ergebnis zeigt Abbildung 2.2:

Abbildung 2.2
Einfache abstrakte Darstellung der Beteiligten

Nun haben wir die Kunden, die Aufträge sowie die Artikel der jeweiligen Aufträge grafisch dargestellt. Betrachten Sie die Grafik, fällt sofort auf: Man könnte diese gruppieren wie in Abbildung 2.3:

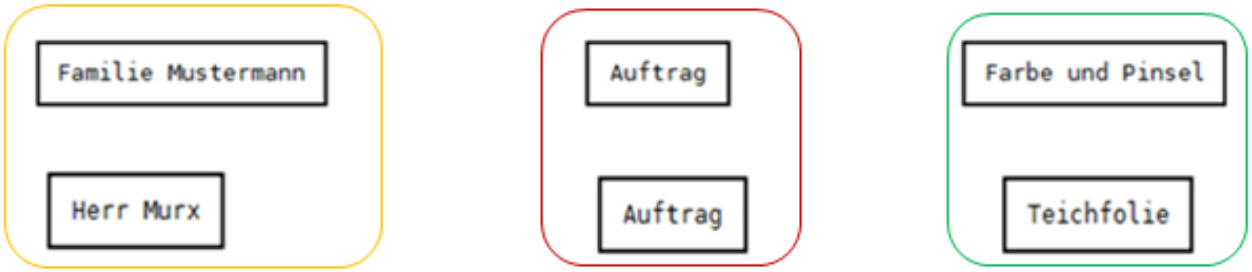


Abbildung 2.3
Gruppierung der Beteiligten

Würde man jetzt noch nach einem schönen passenden Namen suchen, könnte man diesen als Oberbegriff für die einzelnen

Mengen verwenden. Für die erste Menge wäre wohl ”Kunde” ein geeigneter Begriff, für die zweite Menge ”Auftrag” und für die dritte Menge ”Artikel”. In Abbildung 2.4 wurde dies umgesetzt.

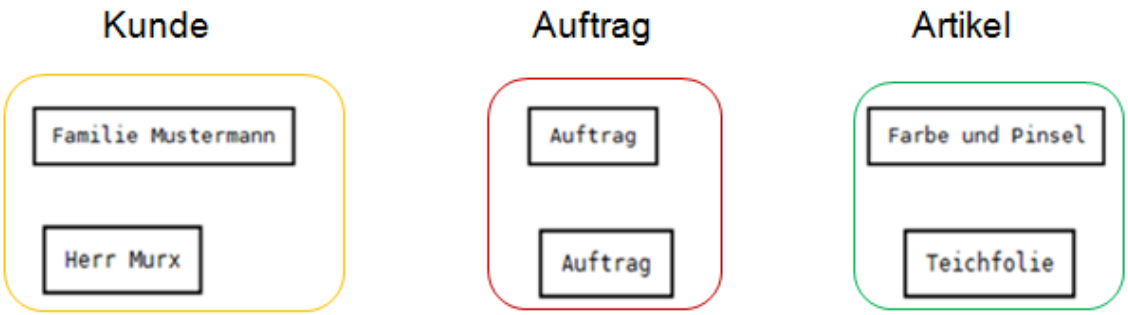


Abbildung 2.4
Gruppierung mit Oberbegriff

Lässt man nun die einzelnen Daten weg und malt ein bisschen, sieht die Grafik folgendermaßen aus (Abb.2.5):

Abbildung 2.5
Darstellung als Entitätstypen

Die Begriffe, welche wir gerade für die verschiedenen Gruppen gewählt haben, bezeichnet man in der ER-Modellierung als Entitätstypen und die einzelnen Kunden, Aufträge sowie Artikel als Entitäten.

2.4 Entitäten

Einzelne Objekte (z.B. Familie Mustermann sowie Herr Murx in unserem Beispiel) werden als Entitäten bezeichnet. Wikipedia definiert Entität wie folgt:

> *"Als Entität (auch Informationsobjekt genannt, englisch entity) wird in der Datenmodellierung ein eindeutig zu bestimmendes Objekt bezeichnet, über das Informationen gespeichert oder verarbeitet werden sollen. Das Objekt kann materiell oder immateriell, konkret oder abstrakt sein. Beispiele: Ein Fahrzeug, ein Konto, eine Person, ein Zustand."*

Das bedeutet eine Entität muss keine natürliche Person sein, oder etwas dass man anfassen kann. Eine Entität kann auch eine Software oder z.B. eine Telefonverbindung sein. Entitäten müssen eindeutig identifizierbar sein. Hat man Entitäten welche nicht eindeutig identifizierbar sind, wie z.B. in dem Beispiel die Entität "Auftrag", so wird eine fortlaufende Nummer vergeben. Mit dieser ist dann z.B. ein einzelner Auftrag eindeutig identifizierbar.

2.5 Entitätstypen

Entitäten kann man häufig in Mengen gleichartiger Objekte unterteilen, z.B. die Menge aller Kunden, die Menge aller Aufträge. Diese Mengen werden als Entitätstypen bezeichnet. Die Entitäten eines Entitätstyps zeichnen sich dadurch aus, dass sie dieselben Eigenschaften haben. Alle Kunden haben die Eigenschaft Name, Straße, Postleitzahl und Ort. Alle Aufträge haben die Eigenschaften Auftragsnummer, Auftragsdatum sowie Art des Auftrags. Es gibt also Entitäten, das sind die einzelnen Subjekte über die wir Daten speichern möchten, und es gibt Entitätstypen, das sind die Obermengen der Entitäten. Schwierig zu verstehen? Dann betrachten Sie bitte Abbildung 2.6:

Kunde ist der Entitätstyp, die einzelnen Kunden sind die Entitäten.
Automarken ist der Entitätstyp, die einzelnen Marken sind die Entitäten.

Weitere Beispiele für Entitäten / Entitätstypen

Entität	**Entitätstyp**
mein Schreibtischstuhl	Stuhl, Möbel, Einrichtung
mein Computer	Computer, Hardware
mein Hund	Hund, Haustier, Familienangehörige

Tabelle 2.2
Beispiele für Entitäten und Entitätstypen

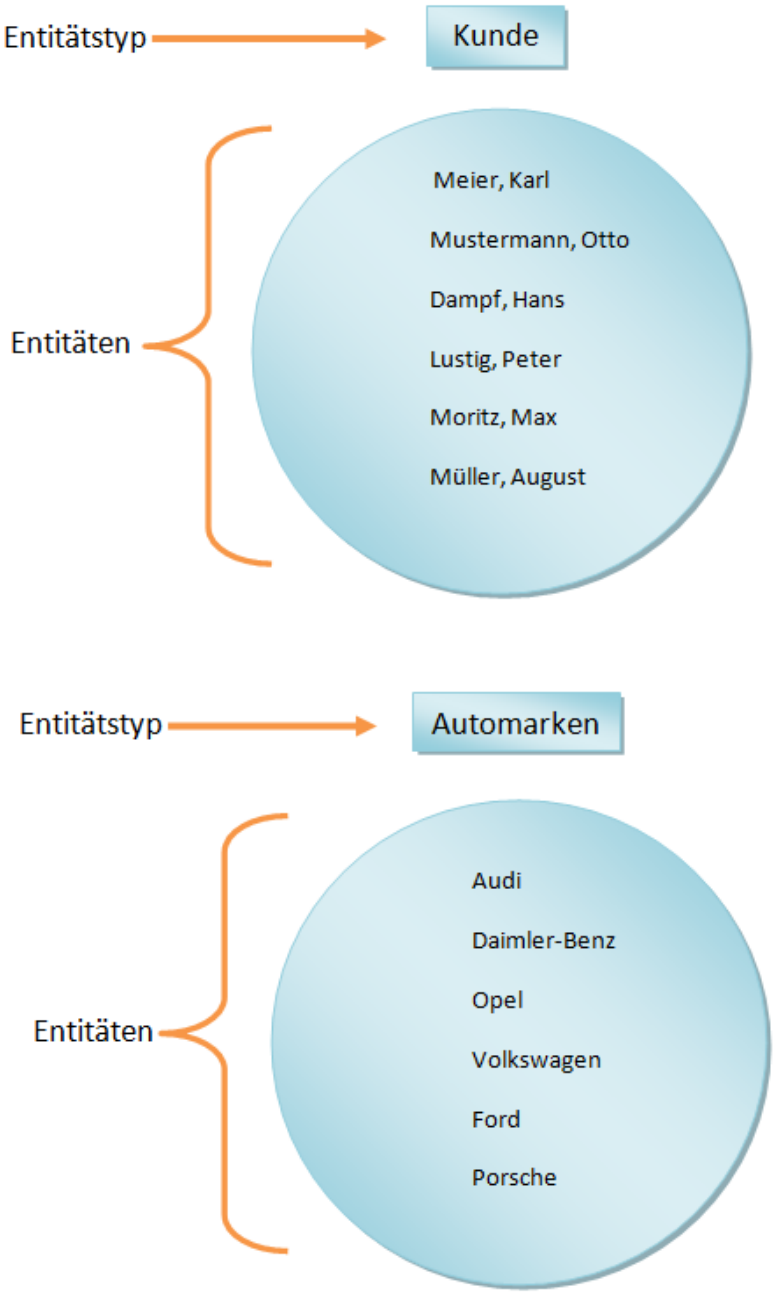


Abbildung 2.6
Veranschaulichung Entitätstypen - Entitäten

Nochmal, um den Unterschied zwischen Entitäten und Entitätstypen zu verdeutlichen:

- Die Entitäten sind die einzelnen Dinge, über die wir Informationen speichern wollen.
- Entitätstypen beschreiben Gruppen gleichartiger Entitäten.

Unterschiedliche Entitäten können übrigens die gleichen Eigenschaften haben, zum Beispiel kann es zwei Kunden mit dem Namen ”Karl Meier” geben. Aber auch wenn wir dieselben Daten speichern, handelt es sich um zwei unterschiedliche Menschen (mit zwei eigenen Identitäten), und damit um zwei verschiedene Entitäten (Abb 2.7).

KdNr	Vorname	Nachname	Strasse	Stadt	Alter
1	Karl	Meier	Lennershofstr	Bochum	45
2	Karl	Meier	Königsallee	Bochum	23

Abbildung 2.7
Verschiedene Entitäten aber gleiche Eigenschaften

2.6 Sonderfälle bei der Suche nach Entitätstypen

Den Text “Ein Kunde kann Aufträge erteilen“ haben wir erfolgreich auf Entitätstypen untersucht. Kunde sowie Auftrag lauteten die Entitätstypen. Das war einfach. Untersuchen Sie bitte nun folgenden Text nach Entitätstypen:
”Ein Auftrag kann Artikel enthalten. Ein Artikel ist z.B. ein Pinsel“
Pinsel ist keine Menge, sondern wiederum nur ”Artikel”. Pinsel ist eine Entität von Artikel.
Anders würde es sich hierbei verhalten:
“Eine Bestellung kann Waren enthalten. Waren sind unterteilt in graue Ware und weiße Ware”.
Hier tritt ein spezieller Fall auf: Waren ist ein Entitätstyp, aber es gibt zwei weitere Entitätstypen: graue sowie weiße Ware. Somit ist Ware der generelle Entitätstyp und graue sowie weiße Ware sind die spezielleren Entitätstypen von dem Entitätstyp Ware.

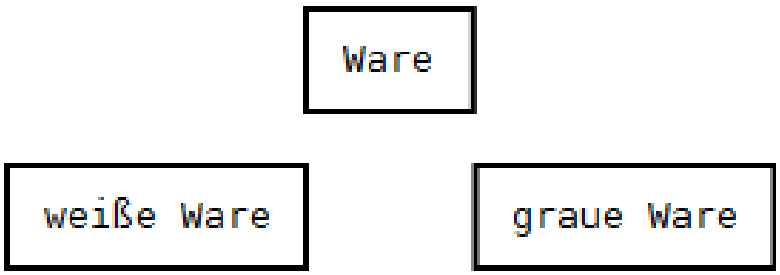


Abbildung 2.8
Generalisierung - Spezialisierung

Hingegen ist hier keine Generalisierung/Spezialisierung erkennbar: “Ski werden in Skitypen eingruppiert. Dies sind z.B. Slalomski, Abfahrtski, Langlaufski” Es gibt hier nur die Mengen ”Ski” sowie ”Skityp”. Slalom-, Abfahrt- sowie Langlaufski sind Skitypen welche dann als Einträge in der Tabelle stehen. Keinesfalls sind das Untermengen von Skityp.

2.7 Unterschied Entität - Attribut

Per Definition sind Entitäten eindeutig zu bestimmende Objekte. Aber was sind dann Attribute und worin besteht der Unterschied zwischen Entitäten und Attributen?
Definition Attribut wikipedia.de:

> *“Typisierung gleichartiger Eigenschaften, z. B. Nachname, Vorname und Eintrittsdatum für den Entitätstyp Angestellter. Das Attribut oder die Attributkombination, deren Wert(e) die Entität eindeutig beschreiben, d.h. diese identifizieren, heißen identifizierende(s) Attribut(e), zum Beispiel ist das Attribut Projektnummer identifizierend für den Entitätstyp Projekt.”*

Analog bedeutet dies z.B. für einen Kunden: Herr Meier ist 46 Jahre alt, 1.69m groß und 65kg schwer. Herr Müller ist 26 Jahre alt, 1.85m groß und 90kg schwer. Die Alters-, Größe sowie Gewichtsangabe sind Attribute von Herrn Meier bzw. Herrn Müller. Als identifizierendes Attribut eignet sich keins der in dem Text aufgeführten Attribute, weil diese nicht eindeutig sind. 46 Jahre alt bzw. 1.69m groß usw. könnte auch ein weiterer Kunde sein. Wie es der Zufall will heißt der

andere Kunde mit den identischen Größen-, Gewichts und Altersangaben auch Herr Meier. Wie kann man nun die beiden Meier eindeutig identifizieren? Mit einem künstlichen Attribut, z.B. einer Kundennummer. Nun fällt es Ihnen möglicherweise wieder wie Schuppen von den Augen, Kundennummer, den Begriff kennen Sie aus Bestellungen und Aufträgen welche Sie bereits getätigt haben. Sie liegen natürlich richtig mit Ihrer Vermutung: Mit Ihrer Kundennummer können Sie eindeutig identifiziert werden, deswegen müssen Sie diese auch immer bei Bestellungen, Rücksendungen oder Reklamationen angeben. Ihre Kundennummer ist natürlich von Unternehmen zu Unternehmen unterschiedlich. Eine Entität ist eine Person, Sache oder Vorgang über den wir Daten speichern möchten. Attribute sind Eigenschaften der Entität, wie z.B. Größe, Haarfarbe, Einkommen, Urlaubstage usw.. Die Attribute sind nicht immer gleich, sie sind auch abhängig von der konkreten Aufgabenstellung. Benötigt werden nur diejenigen Attribute, welche relevant sind. Hat z.B. ein Malerbetrieb eine Kundentabelle, dürften ihn die Haarfarbe seiner Kunden weniger interessieren als z.B. ein Friseurbetrieb, welcher die Haarfarbe ggf. für eine Typberatung benötigt. Es ist hier demnach wie bei den Juristen, es kommt immer auf den Einzelfall an. Attribute werden später, bei der Ableitung der Tabellen aus dem ER-Modell zu Spalten. D.h. gibt es ein Attribut “Gewicht“, so bedeutet dies, später in der Tabelle gibt es eine Spalte “Gewicht“. Attribute werden bei uns nicht im ER-Modell modelliert, da unserer Meinung nach das Modell sonst zu unübersichtlich wird und wir die Auffassung vertreten, dass aus diesem Grunde Attribute im Modell nicht dargestellt werden sollten. Es gibt aber in diversen Büchern, sowie auf Webseiten, ER-Modelle bei denen die Attribute modelliert wurden. Diese sind dann nicht falsch, sondern nur eben anders.

2.8 Beispielsaufgaben zur Bestimmung von Entitätstypen

Wenn wir das Datenmodell für eine konkrete Problemstellung modellieren, orientieren wir uns immer an den Entitätstypen. Die Entitätstypen werden im Modell mit einem Rechteck dargestellt. Auch ohne detaillierte Kenntnisse in der ER-Modellierung kann jetzt schon vermutet werden, dass später aus jedem Entitätstyp eine Tabelle entsteht. Jede Entität ergibt dann eine Zeile und jedes Attribut eine Spalte in der Tabelle. Die Suche nach Entitätstypen kann bei realistischen Problemen recht schwierig sein. Insbesondere die Festlegung, was ist Entitätstyp und was ist Attribut ergibt sich oft nur aus der konkreten Aufgabenstellung. Modellierung ist kein Verfahren, das ein eindeutiges Ergebnis erzeugt - mehrere gleichwertige oder eben auch unterschiedliche gute Lösungen sind möglich. Um gute Modelle von schlechten Modellen unterscheiden zu können, ist ein wenig Erfahrung und Übung erforderlich. Generell gibt es aber ein Vorgehen: Wenn Sie eine Aufgabenstellung sehen, deuten die Substantive dieser Aufgabenstellung meistens auf Entitäten hin. Sammeln Sie also zunächst die Substantive, und prüfen Sie dann, ob sich jeder dieser Begriffe als Entität eignet, und ob es vielleicht noch weitere Entitäten geben müsste, um die Problemstellung lösen zu können. Nachfolgend ein paar Beispiele zur Identifikation von Entitätstypen.

2.8.1 Beispiel 1: Mitarbeiter

- Mitarbeiter arbeiten in Abteilungen
- Abteilungen werden von Mitarbeitern geleitet
- Mitarbeiter können mit Mitarbeitern verheiratet sein

In diesem Beispiel sind nur zwei Entitätstypen vorhanden: Mitarbeiter und Abteilungen.

Abbildung 2.9
Entitätstypen des Beispiels Mitarbeiter

2.8.2 Beispiel 2: Vorlesungsplan

In den Veranstaltungsräumen der Hochschule sind Ressourcen installiert. Als Ressourcen werden Beamer, Tafeln, Computer usw. bezeichnet. Räume besitzen darüber hinaus eine Kapazitätsklasse. Damit ist eine Klassifizierung in Bezug auf

ihre Anzahl an Sitzplätzen gemeint. Professoren der Hochschule führen Veranstaltungen in festgelegten Zeitblöcken an Wochentagen in den Vorlesungsräumen durch. Die Benutzer des Informationssystems sind die Professoren und StudentInnen. Hier lauten die Entitätstypen:

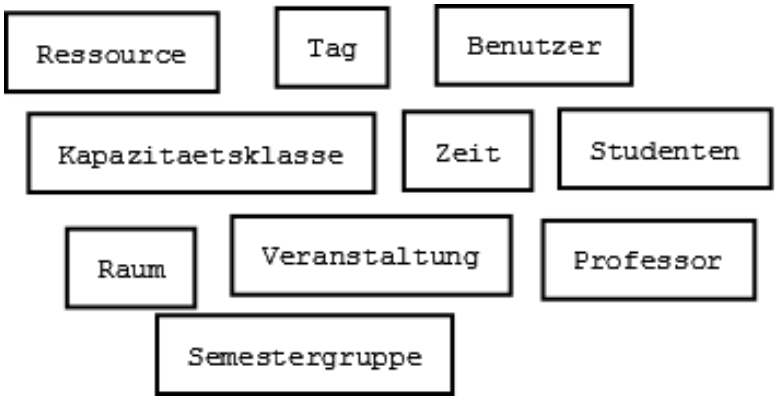


Abbildung 2.10
Entitätstypen des Beispiels Vorlesungsplan

Wir haben nun auf den letzten Seiten die Entitätstypen analysiert und grafisch dargestellt. Die Darstellung ist aber noch recht unübersichtlich. Beim Beispiel ”Vorlesungsplan” sehen wir zehn Entitätstypen, welche später (bei der Ableitung) jeweils eine eigene Tabelle ergeben. In der Einführung haben Sie gelernt, dass man die Tabellen irgendwie “verknüpfen“ muss. Dies ist erforderlich damit man auch die Beziehung, welche zwei Entitätstypen miteinander haben, darstellen kann.

2.9 Darstellung von Beziehungen

ERM bedeutet Entity-Relationship-Modelling. Das ”E” welches für Entity steht haben wir nun abgearbeitet. Kommen wir nun zum “R“ welches für Relationship, also Beziehung steht. Wir untersuchen nun ob es zwischen Entitätstypen Beziehungen gibt. Diese Beziehungen sind äußerst wichtig, da diese anschließend in der Datenbank abgespeichert werden müssen. Beziehungen werden gefunden indem man den Text auf Verben untersucht.

Beispiel aus der Aufgabe Vorlesungsplan:
Räume besitzen darüber hinaus eine Kapazitätsklasse Das bedeutet zwischen Raum sowie Kapazitätsklasse existiert eine Beziehung. Diese muss grafisch, wie nachfolgend dargestellt, modelliert werden.

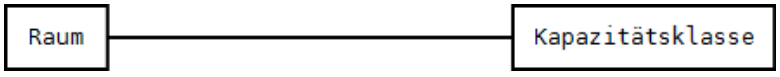


Abbildung 2.11
Vorlesungsplan: Beziehung Raum - Kapazitätsklasse

Zwischen den beiden Entitätstypen ”Raum” und ”Kapazitätsklasse” wird eine einfache, durchgezogene Line gezeichnet. Diese zeigt, dass zwischen den beiden Entitätstypen eine Beziehung besteht.

Beispiel aus Aufgabe Mitarbeiter:
Mitarbeiter arbeiten in Abteilungen
Die zwei Entitätstypen “Mitarbeiter“ und “Abteilung“ werden ebenso durch eine einfache Linie miteinander verbunden (Abb. 2.12). Wir wissen nun, dass zwischen diesen beiden Entitätstypen eine Beziehung existiert.
Wir untersuchen jetzt das Beispiel Vorlesungsplan auf alle möglichen Beziehungen der Entitätstypen untereinander. Wir gehen dabei satzweise vor.

1. In den Veranstaltungsräumen der Hochschule sind Ressourcen installiert. Raum hat eine Beziehung zu Ressource, umgekehrt hat natürlich auch Ressource eine Beziehung zu Raum. Hochschule ist kein Entitätstyp. Sicherlich wäre

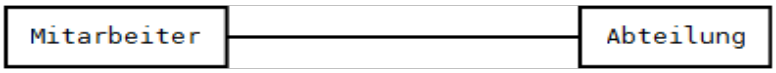


Abbildung 2.12
Beziehung Mitarbeiter Abteilung

es möglich eine Beziehung zwischen Hochschule und Veranstaltungsraum zu definieren, in etwa ”Eine Hochschule hat mehrere Veranstaltungsräume”. Da sich der Vorlesungsplan jedoch auf nur eine Hochschule bezieht und in dem Aufgabentext nichts davon steht dass wir den Vorlesungsplan von verschiedenen Hochschulen modellieren sollen, gibt es in dieser Aufgabe keinen Entitätstyp “Hochschule“. Wir haben übrigens die Bezeichnung “Raum“ statt “Veranstaltungsraum” gewählt, weil dieser Begriff kürzer ist und wir schreibfaul sind.

2. Räume besitzen darüber hinaus eine Kapazitätsklasse. Raum hat eine Beziehung zu Kapazitätsklasse, umgekehrt hat Kapazitätsklasse auch eine Beziehung zu Raum.

3. Professoren der Hochschule führen Veranstaltungen in festgelegten Zeitblöcken an Wochentagen in den Vorlesungsräumen durch.

Dies darzustellen ist nicht ganz so trivial wie die vorhergehenden Aufgaben, aber natürlich lösbar. Es ist zu beachten, dass hier sechs Entitätstypen eine Beziehung darstellen:

Entitätstyp	**Beschreibung**
Professor	sind die Lehrenden, ohne sie keine Veranstaltung
Veranstaltung	das sind die Vorlesungen, wie z.B. Einführung in BWL
Zeit	Der Zeitblock in welchem die Veranstaltung statt findet, z.B. 8-10 Uhr
Raum	zeigt in welchem Raum die Veranstaltung durchgeführt wird
Kapazitätsklasse	Einteilung der Hörsäle nach Anzahl Sitzplätze, z.B. 40-60 Plätze, 61-80 Plätze, 81-100 Plätze
Tag	Tag an welchem die Veranstaltung durchgeführt wird
Semestergruppe	zeigt welches Semester die Veranstaltung hören sollte

Tabelle 2.3
Beteiligte am Vorlesungsplan

Diese Konstellation unterscheidet sich von den vorherigen dadurch, dass hier mehr als zwei Entitätstypen beteiligt sind. Daher weicht auch die grafische Darstellung in Abbildung 2.13 ab:

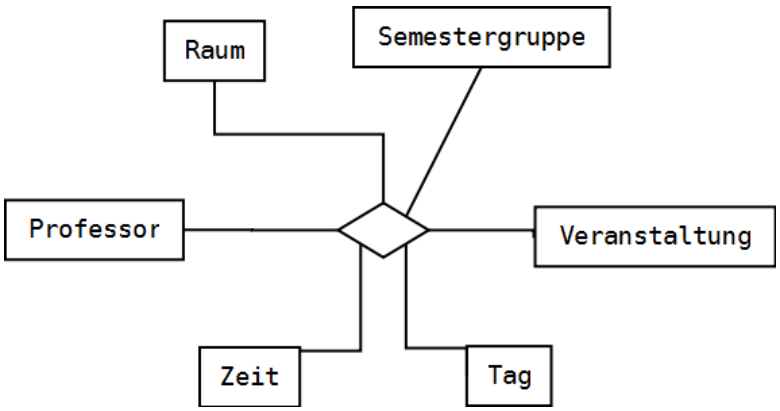


Abbildung 2.13
Beziehungen Stundenplan

2.10 Kardinalitäten von Beziehungen

Wir sind jetzt schon ein ganzes Stück näher an unserer Datenbank. Wir haben Entitätstypen gefunden und deren Beziehungen zueinander modelliert. Jetzt fehlt noch ein Schritt und dann ist unser Modell fertig: Die Kardinalitäten. Eine Kardinalität legt fest mit wie vielen Entitäten eine gegebene Entität minimal sowie maximal in Beziehung stehen kann. Hört sich kompliziert an, ist es aber nicht, wie das Beispiel in Abbildung 2.14 zeigt:

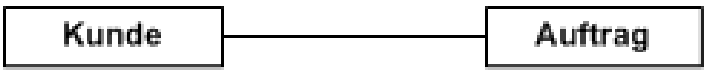


Abbildung 2.14
Beziehung Kunde Auftrag

Die Abbildung 2.14 können Sie bereits lesen. Sie sagt aus, dass zwischen Kunde und Auftrag eine Beziehung besteht. Nun untersuchen wir die Kardinalität von Kunde zu Auftrag. Dies machen wir indem wir fragen:
Wie viele Aufträge kann ein Kunde erteilen? Wir sind der Meinung, ein Kunde kann entweder keinen bis beliebig viele Aufträge erteilen. Somit lautet der Satz:
“Ein Kunde kann keinen bis beliebig viele Aufträge erteilen”.
Die Leserichtung ist von Kunde zu Auftrag. Jetzt drehen wir die Leserichtung um und fragen:
Ein Auftrag, muss der immer von einem Kunden kommen?
Wir behaupten: Ja, ein Auftrag muss immer von einem Kunden kommen (von wem denn sonst?). Das “muss“ in dem Satz ist sehr wichtig, weil es aussagt, dass es keinen Auftrag gibt welcher nicht von einem Kunden kommt. Das bedeutet es gibt keinen Auftrag welcher nicht einem Kunden zugeordnet werden kann. Zum besseren Verständnis betrachten Sie bitte die Abbildung 2.15.
Sie erkennen, dass es Kunden gibt, die keinen Auftrag erteilt haben. Ebenso gibt es Kunden die gleich mehrere Aufträge erteilt haben. Dies Konstellation, wenn es Kunden gibt die keinen Auftrag erteilt haben und es aber auch Kunden gibt die mehrere Aufträge erteilt haben wird als ”0..*” Kardinalität notiert. Ausgesprochen wird das ”mindestens 0, maximal beliebig viele“.
Wir haben nun die Beziehung Kunde Auftrag einmal mit der Leserichtung Kunde zu Auftrag und einmal mit der Leserichtung Auftrag zu Kunde geprüft. Wir zeichnen die Kardinalität in die Grafik ein. Die Abbildung 2.17 zeigt das fertige Modell.

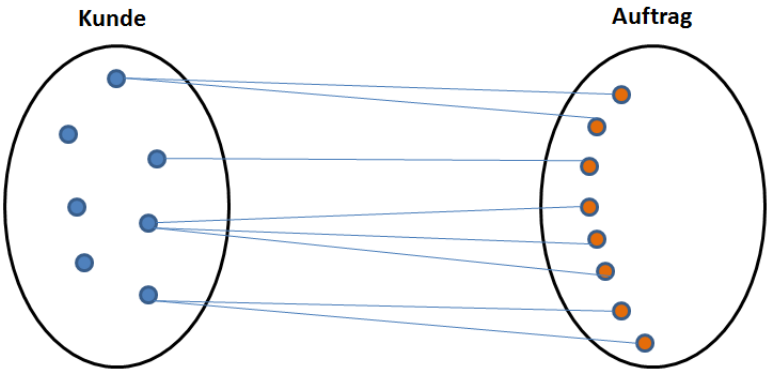


Abbildung 2.15
Kardinalitaeten in der Beziehung Kunde Auftrag

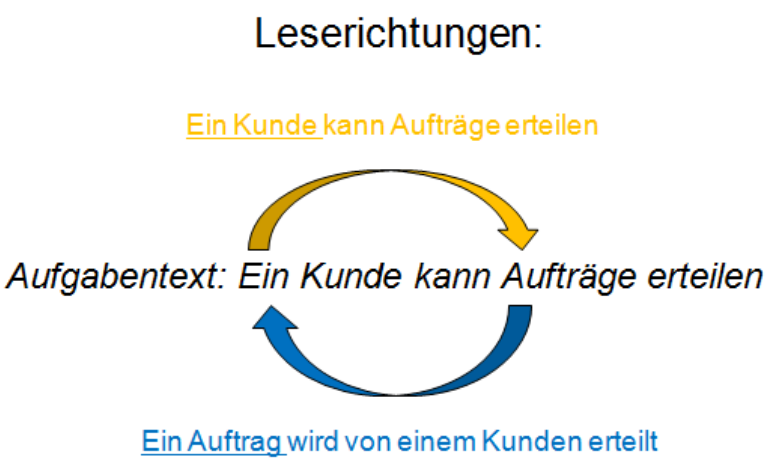


Abbildung 2.16
Leserichtungen

Abbildung 2.17
Leserichtungen

2.11 Beziehungstypen

2.11.1 Rekursive Beziehung

Bei einer rekursiven Beziehung stehen zwei Entitäten desselben Entitätstyps miteinander in Beziehung. Beispiel: Es gibt ein Buch in verschiedenen Auflagen. Das bedeutet die 5. Auflage des Buches ist die Vorgängerversion der 6. Auflage. Oder anders ausgedrückt: Die 6. Auflage ist die Nachfolgeauflage der 5. Auflage. Die Beziehung zwischen diesen beiden Büchern wird als rekursive Beziehung bezeichnet. Rekursive Beziehungen sind auf zwei Arten grafisch darstellbar. Abbildung 2.18 und 2.19 zeigen Ihnen die beiden grafischen Darstellungsarten.

Kardinalität	Bedeutung
0..1	maximal 1
1..1	genau 1 (darf auch ”1” geschrieben werden)
1..*	mindestens 1, maximal beliebig viele
0..*	mindestens 0, maximal beliebig viele (kann auch “*“ geschrieben werden)

Tabelle 2.4
Übersicht Kardinalitäten

Angewandte Kryptographie	5. Auflage	3.Juni 2015	ISBN: 3893198547
Angewandte Kryptographie	6. Auflage	19.Dezember 2016	ISBN: 3827372283

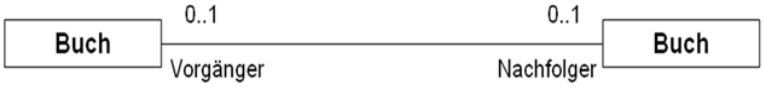


Abbildung 2.18
Erste Darstellungsart Vorgänger-Nachfolger

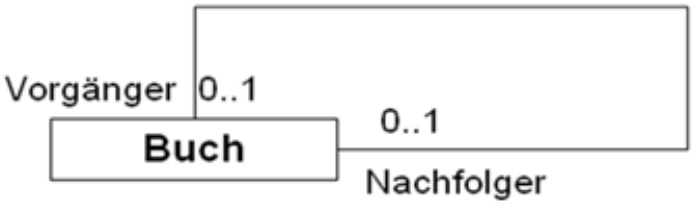


Abbildung 2.19
Zweite Darstellungsart Vorgänger-Nachfolger

Bei rekursiven Beziehungen treten die beiden beteiligten Entitäten in unterschiedlichen Rollen auf. Es sind zwar beides Bücher, aber in dieser speziellen Beziehung ist das eine Buch in der Rolle des Vorgängers, das andere Buch in der Rolle des Nachfolgers. Damit klar wird, welche Rollen gemeint sind, können die Rollen an die Enden der Beziehung geschrieben werden. Dies ist besonders dann hilfreich, wenn die Kardinalitäten an den beiden Enden der Beziehung unterschiedlich sind. Sie sehen hier, dass man rekursive Beziehungen auf zwei Arten darstellen kann.

- Der Entitätstyp wird zweimal in Form von Rechtecken gemalt, dann wird die Beziehung wie immer als Linie zwischen den beiden Rechtecken gezeichnet.
- Der Entitätstyp wird nur einmal gezeichnet, dann beginnt und endet die Beziehung bei demselben Rechteck.

Beispiel 2 zu rekursiven Beziehung
Annahme: Mitarbeiter ist verheiratet mit Mitarbeiter.

2.11.2 1:1 Beziehungen

Ein Mitarbeiter kann eine Abteilung leiten, eine Abteilung wird von einem Mitarbeiter geleitet (Abb. 2.21). Dies ist eine klassische 1:1 Beziehung. Sie beschreibt, dass ein Mitarbeiter eine Abteilung leiten kann und eine Abteilung von einem Mitarbeiter geleitet werden muss. Eine Seite ist demnach eine Muss-Beziehung (Abteilung muss geleitet werden), die andere Seite eine Kann-Beziehung (Mitarbeiter kann Abteilung leiten). Das erkennt man daran, dass 0..*, also mindestens

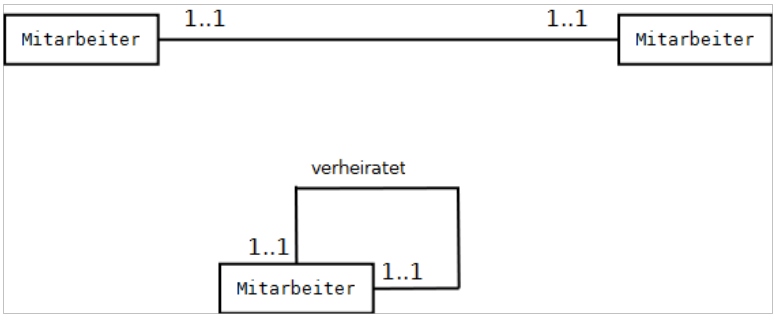


Abbildung 2.20
Rekursive Beziehung Mitarbeiter ist verheiratet mit Mitarbeiter

0, maximal beliebig viele als Kardinalität verwendet wird. Immer wenn Sie die Notation 0..* lesen, ist diese Beziehung optional. Anders sieht es aus wenn Sie 1..* auf einer Seite lesen, dann ist diese Beziehung nicht optional. Es ist eine Muss-Beziehung welche immer eintritt (Abb 2.21).

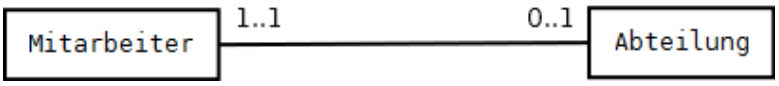


Abbildung 2.21
1:1 Kardinalität Mitarbeiter Abteilung

Bei der 1:1 Beziehung gibt es noch die beidseitige Kann-Beziehung sowie die beidseitige Muss-Beziehung. Jedoch sind diese für die Vorlesung nicht relevant und werden daher in diesem Script nur der Vollständigkeit wegen erwähnt.

2.11.3 1:N Beziehung

1:N Beziehungen unterscheiden sich von den 1:1 Beziehungen dadurch, dass hier auf einer Seite mehrere Entitäten auftreten können. 1:N-Beziehungen sind die häufigsten Beziehungen bei der Modellierung von Datenbanken. Bei den 1:N-Beziehungen unterscheidet man zwei Ausprägungen: optionale 1:N Beziehung oder nicht optionale 1:N-Beziehung wie nachfolgende Abbildung 2.22 verdeutlicht.

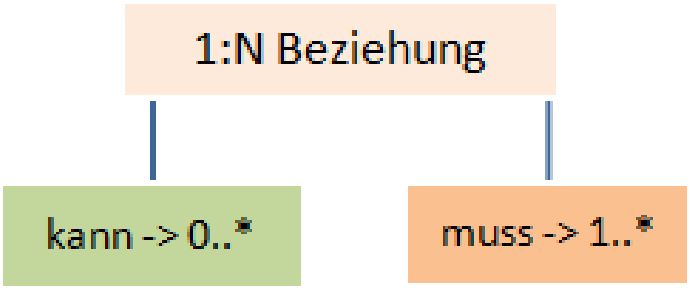


Abbildung 2.22
Übersicht der 1:N Beziehungen

Optionale 1:N Beziehung
Eine optionale 1:N Beziehung ist beispielweise: Ein Bürger kann Wohnungseigentum besitzen. Ein Haus oder eine Wohnung muss immer einen Bürger als Besitzer haben (Abb. 2.23).

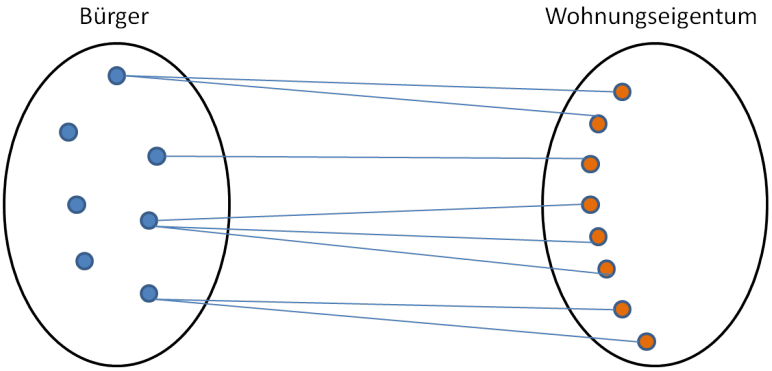


Abbildung 2.23
Beziehung von Bürger zu Wohnungseigentum

Beim Betrachten der Abbildung 2.23 fällt auf, dass es Bürger gibt welche kein Wohnungseigentum besitzen. Demnach ist die Beziehung ”Bürger-Wohnungseigentum” optional, eine Kann-Beziehung. Das Modell für diese Aufgabenstellung sehen Sie in Abbildung 2.24:

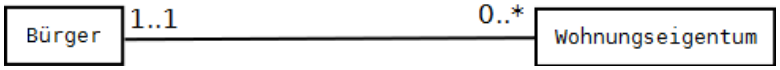


Abbildung 2.24
Kardinalität Bürger - Wohnungseigentum

Weiteres Beispiel für eine optionale 1:N Beziehung:
Ein Kunde kann verschiedene Aufträge erteilen. Ein Auftrag kommt immer von einem Kunden (Abb.2.25)

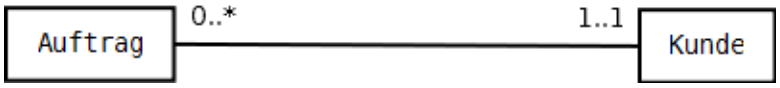


Abbildung 2.25
Kardinalität Auftrag - Kunde

Dieses ER-Modell liest man:
Ein Kunde kann keine (Null) bis beliebig viele Aufträge erteilen. Ein Auftrag kommt immer von mindestens einem und maximal einem Kunden.
Diese Beziehung bezeichnet man als “1:N kann“. Das kann bedeutet diese Beziehung ist optional, sie muss nicht eintreten. Erteilt ein Kunde keinen Auftrag, so tritt diese Beziehung nicht ein und wird demnach auch nicht abgespeichert. Ob eine Beziehung eintreten ”kann” oder eintreten ”muss” ist für die später folgende Ableitung der Tabellen relevant. Dazu später mehr im Kapitel “Tabellen ableiten”.

Nicht optionale 1:N Beziehung
Schauen wir uns die Beziehung zwischen einer Mutter und ihren Kindern in Abbildung 2.26 an:

- Eine Mutter kann eins oder mehrere Kinder haben.
- Zu einem Kind gehört immer eine Mutter, Kinder ohne Mutter gibt es nicht.

Eine Mutter hat eins oder mehrere Kinder, ein Kind hat immer eine Mutter. Daraus ergibt sich folgendes ER-Modell in Abbildung 2.27:

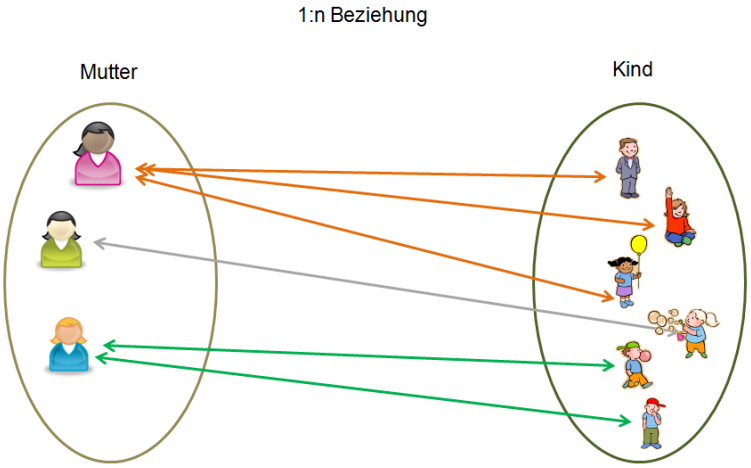


Abbildung 2.26
Kardinalität Mutter Kind

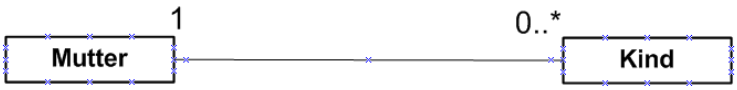


Abbildung 2.27
Kardinalität Mutter Kind

1. Leserichtung: Mutter hat Kind

Zu einer Mutter gehört

- Mindestens 1 Kind
- (theoretisch) Maximal beliebig viele Kinder

2. Leserichtung: Ein Kind hat immer eine Mutter

Zu einem Kind gehören

- Mindestens 1 Mutter
- Maximal 1 Mutter

Sie sehen hier eine 1:N-Muss Beziehung, da ein Kind immer eine Mutter haben muss. Umgekehrt muss eine Mutter immer ein Kind haben, sonst wäre die Frau keine Mutter.

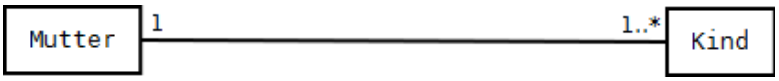


Abbildung 2.28
Mutter-Kind Beziehung

2.11.4 m:n Beziehung

Eine m:n Beziehung wird “many to many“ ausgesprochen und wird benutzt wenn mehrere Entitäten auf der linken Seite mit mehreren Entitäten auf der rechten Seite in Beziehung stehen können. In unserem Beispiel kann ein Auftrag mehrere Artikel enthalten und ein Artikel kann in mehreren Aufträgen enthalten sein.

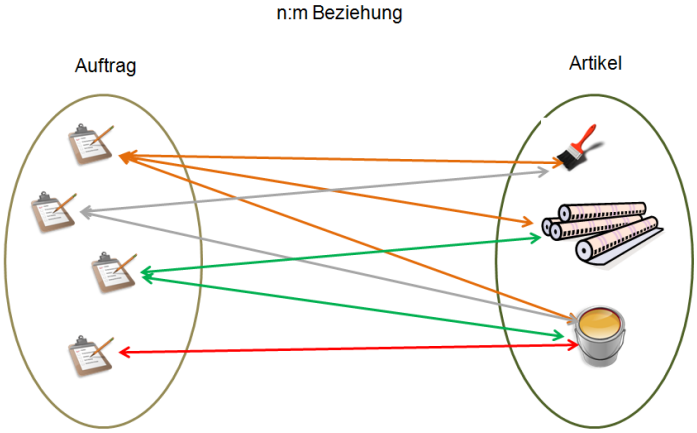


Abbildung 2.29
Darstellung m:n Beziehung

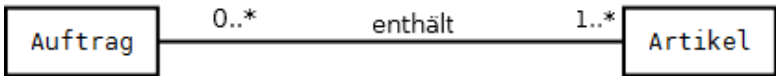


Abbildung 2.30
Darstellung einer m:n Beziehung im ER-Modell

Diese m:n Beziehung stellt folgendes dar:
Eine Bestellung enthält Artikel, mindestens einen, maximal beliebig viele.
Ein Artikel kann in keiner oder beliebig vielen Bestellungen vorhanden sein.
Beachten Sie: Die Entität, von der die Betrachtung ausgeht, wird immer im Singular verwendet:
Ein Auftrag kann mehrere Artikel enthalten, ein Artikel kann in mehreren Aufträgen vorkommen.
Ein Kunde kann mehrere Aufträge erteilen, ein Auftrag kommt von einem Kunden.
Das ist aber auch einleuchtend, denn würden wir diese Entität im Plural verwenden, erhielten wir nur n:m-Beziehungen in unseren Modellen. Denn: Mehrere Aufträge können selbstverständlich von mehreren Kunden kommen, mehrere Räume können natürlich mehrere Kapazitätsklassen besitzen. Das würde natürlich den kreativen Prozess der Modellbildung stark vereinfachen :-), aber leider auch ziemlich sinnlos machen. Betrachten wir auch hier ein zweites Beispiel, die Beziehung zwischen Raum und Ressource aus CampusInfo. Hier gilt: Ein Raum kann mehrere Ressourcen besitzen (z.B. Beamer, Tafel und Overhead-Projektor), eine Ressource kann es in mehreren Räumen geben (wir haben eine Menge Beamer-Räume, eine Tafel gibt es in jedem Vorlesungsraum).

2.11.5 Beziehungen zwischen mehr als zwei Entitätstypen

Sie erinnern sich an folgende Grafik ein paar Seiten vorher?
Die Abbildung 2.31 stellt bereits eine m:n Beziehung dar. Hier sind mehr als zwei Entitätstypen beteiligt. Die Beziehung wird als Beziehung 6. Grades bezeichnet, weil eben sechs Entitätstypen an der Beziehung beteiligt sind.

Eine Beziehung größer 2. Grades wird immer in einer neuen Tabelle abgebildet. In dieser sind die Primärschlüssel der Entitätstypentabellen zusammengesetzt der neue Primärschlüssel. Zusätzliche Attribute sind möglich.

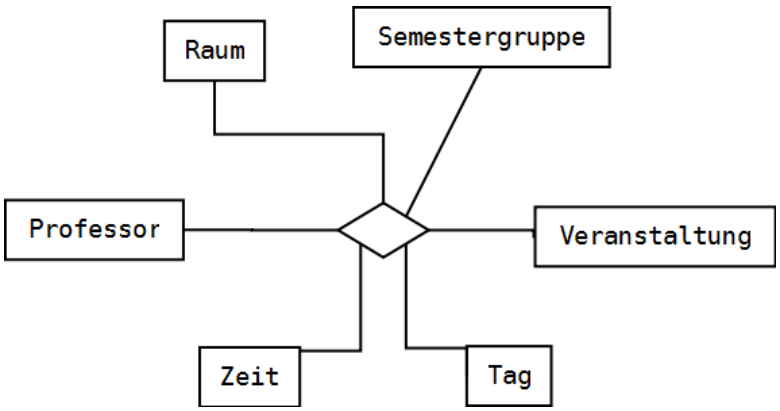


Abbildung 2.31
Grafische Darstellung einer Beziehung 6. Grades

Weiteres Beispiel für eine Beziehung vom Grad > 2:
“Werbung erscheint in einem Zeitraum in einem OnlineArtikel“ Dieser Satz beschreibt, dass eine Werbung, z.B. für ein BWL-Studium in einem Zeitraum (angenommen Mitte April bis Mitte Mai) in einem OnlineArtikel erscheinen kann. Die Aufgabe beinhaltet drei Entitätstypen welche zueinander in Beziehung stehen. Man spricht in diesem Zusammenhang auch von einer Beziehung dritten Grades (weil drei Entitätstypen daran beteiligt sind). Diese drei Entitätstypen stehen alle zueinander in Beziehung und diese Beziehungen werden folgendermaßen grafisch dargestellt:

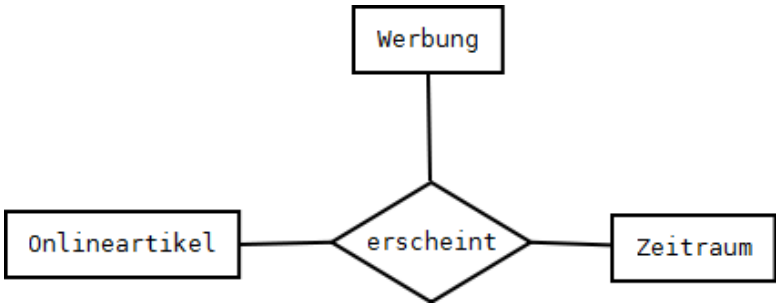


Abbildung 2.32
Entitätstypen OnlineArtikel

Beziehungen vom Grad > 2 zu entdecken, ist relativ schwierig (zumindest schwieriger, als Grad = 2). Denken Sie immer dann über solche Beziehungen nach, wenn in der Aufgabenstellung in einem Satz mehr als zwei Entitäten von einem einzelnen Verb betroffen sind. Für Beziehungen vom Grad > 2 ist in der Datenbank-Literatur keine Definition dieser Bedeutung der Kardinalität solcher Beziehungen zu finden. In älteren Skripten und Folien sind manchmal dennoch Kardinalitäten zu sehen - ignorieren Sie diese einfach, und gehen Sie immer davon aus, dass generell beliebig viele Beziehungen zwischen den betroffenen Entitäten existieren können.

2.11.6 Generalisierung - Spezialisierung

Die letzte Beziehung, die wir abbilden, ist die Generalisierung - Spezialisierung. Dies machen wir uns wieder am Beispiel unseres CampusInfo-Systems klar: Dieses verfügt über zwei Arten von Benutzern: Professoren sowie Studenten. Diese Benutzer haben viel gemeinsam, nämlich Namen, Vornamen, E-Mail-Adresse, Handynummer und so weiter. Sie unterscheiden sich aber auch. Studenten haben eine Matrikelnummer, einen Studiengang, Anzahl Semester, dies sind alles keine Eigenschaften von Professoren. Professoren wiederum verfügen über ein Büro an der Hochschule, haben eine Sprechstunde, eine dienstliche Telefonnummer, ein Lehrgebiet, Das sind alles keine Eigenschaften von Studierenden. Wir sagen, es gibt eine Generalisierung, (die Benutzer) mit zwei Spezialisierungen, (den Professoren und Studenten).

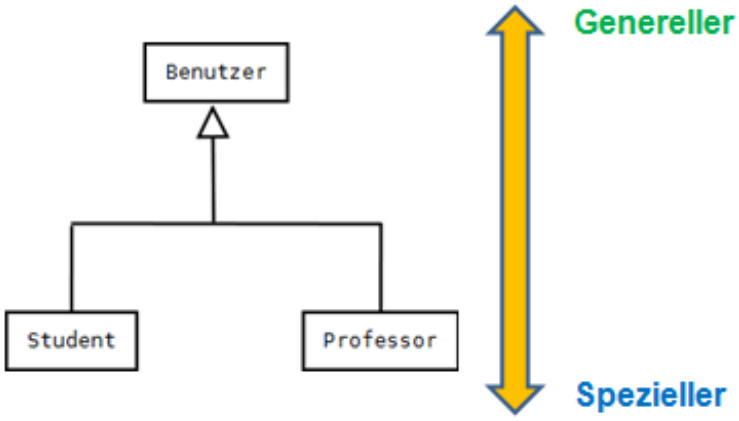


Abbildung 2.33
Generalisierung / Spezialisierung

Weiteres Beispiel:

Artikel können nicht nur Bücher, sondern auch DVDs (also Filme) oder CDs (Musik) sein. Manchmal kommt es vor, dass verschiedene Entitäten sowohl gemeinsame als auch unterschiedliche Eigenschaften haben können. So können Sie in dem Versand sowohl Bücher, als auch Filme oder CDs bestellen. Die gemeinsamen Eigenschaften sind zum Beispiel die Tatsache, dass wir diese Entitäten bestellen können und, dass sie einen Preis und eine Beschreibung haben. Allerdings unterscheiden sich die Entitäten auch: Bücher haben eine ISBN. Bei Filmen wird im Rahmen der freiwilligen Selbstkontrolle ein Mindestalter festgelegt. Bei CDs wollen wir speichern, welche Musiktitel (von wem gesungen) darauf vorkommen. Mit Hilfe der Generalisierung/Spezialisierung können Gemeinsamkeiten und Unterschiede einfach herausgearbeitet werden. Zunächst wird ein Entitätstyp modelliert, der die Generalisierung darstellt. Dieser repräsentiert Eigenschaften, die alle beteiligten Entitäten gemeinsam haben. Weiterhin werden beliebig viele Spezialisierungen modelliert. Hierdurch können dann die Eigenschaften repräsentiert werden, die nur eine Teilmenge der Entitäten betreffen. Die Eigenschaften einer Entität (also zum Beispiel eines Buches) werden dadurch über verschiedene Entitätstypen verteilt. Man spricht auch gerne von Vererbung: die spezielleren Entitätstypen erben die Eigenschaften der generelleren Entitätstypen. Jede Entität des spezielleren Typs ist auch eine Entität des generelleren (allgemeineren) Typs, d.h., die Menge der Bücher ist eine Teilmenge aller Artikel. Dies gilt genauso für die Filme und die CDs. Also gilt:

- Jedes Buch ist ein Artikel.
- Jede MusikCD ist ein Artikel.
- Jede FilmDVD ist ein Artikel.

Manchmal hilft die Mengenlehre, dieses Konzept besser zu verstehen :-)!

2.11.7 Zusammenfassung

Bei der Modellierung von Datenbanken ist der erste Schritt die Suche nach Entitätstypen. Sind diese gefunden, werden die Beziehungen zwischen den Entitätstypen definiert. Der Eintrag der Kardinalitäten vervollständigt das Modell. Bei den Beziehungen unterscheidet man folgende:

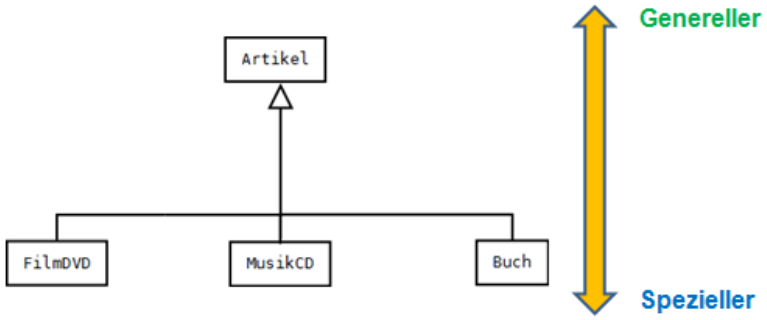


Abbildung 2.34
Generalisierung / Spezialisierung bei Buchversand

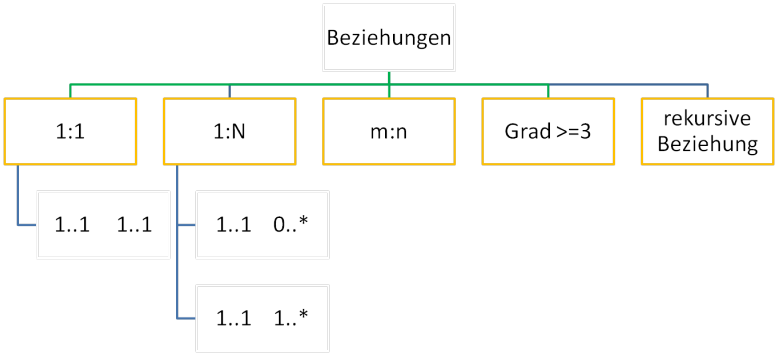


Abbildung 2.35
Übersicht Kardinalitäten

Beziehung	**Beispiel**
1:1	Ein Fahrzeug hat ein Lenkrad, Ein Lenkrad gehört zu einem Fahrzeug
1:N	Ein Student schreibt mehrere Klausuren, eine Klausur wird von einem Student geschrieben
m:n	Eine Bibliothek besitzt mehrere Bücher, ein Buch kann in mehreren Bibliotheken vorhanden sein
Grad >=3	Werbung erscheint in einem Artikel in einem bestimmten Zeitraum
Rekursive Beziehung	Mitarbeiter ist verheiratet mit Mitarbeiter
Generalisierung /Spezialisierung	Mitarbeiter sind unterteilt in Arbeiter und Angestellte

Tabelle 2.5
Beispiele für Beziehungen

Kapitel 3

Ableiten von Tabellen

Wir haben im vorherigen Kapitel das ER-Modell erstellt. Nun kommt die Fleißarbeit: Die Ableitung der Tabellen aus dem Modell. Dies ist sehr einfach, weil man nur ein paar Regeln anwenden muss um die Tabellen erfolgreich abzuleiten. Wir fangen direkt mit einem Beispiel an:

Abbildung 3.1
Das Modell von Bürger - Wohnungseigentum

Das Beispiel kennen Sie bereits. Wir leiten nun die Tabellen mittels der ersten Regel ab.
Diese lautet:
Aus jedem Entitätstyp entsteht eine Tabelle.

Das bedeutet wir können direkt zwei Tabellen anlegen, Bürger sowie Wohnungseigentum. Das ist trivial. Nun müssen wir aber die Beziehung zwischen diesen beiden Entitätstypen abbilden. Möchte man Beziehungen abbilden, so gibt es grundsätzlich nur zwei Möglichkeiten:

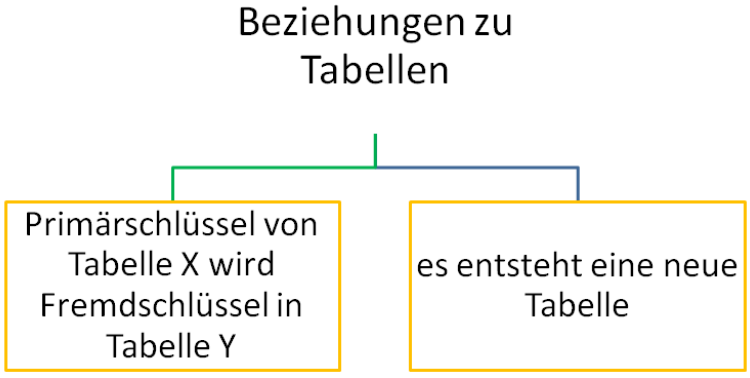


Abbildung 3.2
Möglichkeiten bei der Ableitung

3.1 1:1 Beziehung

Bei einer 1:1 Beziehung hat man freie Auswahl, in welche Tabelle man den Primärschlüssel der anderen Tabelle als Fremdschlüssel einfügt.

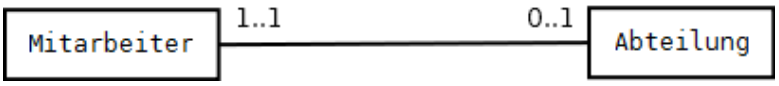


Abbildung 3.3
Mitarbeiter leitet Abteilung Beziehung

Aus diesem Modell entstehen die Tabellen:

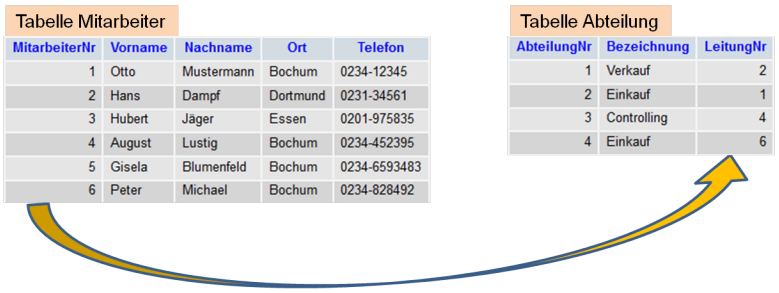


Tabelle Mitarbeiter

MitarbeiterNr	Vorname	Nachname	Ort	Telefon
1	Otto	Mustermann	Bochum	0234-12345
2	Hans	Dampf	Dortmund	0231-34561
3	Hubert	Jäger	Essen	0201-975835
4	August	Lustig	Bochum	0234-452395
5	Gisela	Blumenfeld	Bochum	0234-6593483
6	Peter	Michael	Bochum	0234-828492

Tabelle Abteilung

AbteilungNr	Bezeichnung	LeitungNr
1	Verkauf	2
2	Einkauf	1
3	Controlling	4
4	Einkauf	6

Abbildung 3.4
Ableitung der Beziehung Mitarbeiter leitet Abteilungïn Tabellen

Der Primärschlüssel der Tabelle ”Mitarbeiter” wandert als Fremdschlüssel in die Tabelle “Abteilung”. Hierfür ist die Tabelle “Abteilung“ um eine Spalte LeitungNr zu erweitern. Die Beziehung ”Mitarbeiter leitet Abteilung“ ist optional, weil nicht jeder Mitarbeiter eine Abteilung leitet. Die Beziehung ”Abteilung wird von Mitarbeiter geleitet“ ist eine muss-Beziehung, weil eine Abteilung von einem Mitarbeiter geleitet werden muss.
Wenn Sie sich die Abbildung 3.4 genauer anschauen, werden Sie feststellen: Es wäre auch möglich den Primärschlüssel von Abteilung als Fremdschlüssel in Mitarbeiter einzufügen. Das stimmt grundsätzlich, aber bei angenommenen 10 Abteilungen und 1000 Mitarbeitern hätten Sie dann in der Tabelle Mitarbeiter eine Spalte welche 990 leere Zellen und 10 Zellen mit Werten. Das macht keinen Sinn, es sei denn Sie möchten sehen wie Schweizer Käse aussieht :-). Haben Sie eine optionale 1:1 Beziehung, so wird der Primärschlüssel der 1..1-Beziehung in die Tabelle der 0..1 Beziehung als Fremdschlüssel eingefügt.

3.2 1:N Beziehung

In unserem Beispiel “Bürger besitzt Wohnungseigentum, Wohnungseigentum gehört Bürger“ wird der Primärschlüssel von der Tabelle ”Bürger” Fremdschlüssel in der Tabelle ”Wohnungseigentum“. Es entsteht somit bei einer 1:N-Beziehung keine neue Tabelle. Wir haben uns darauf verständigt, bei der Ableitung von Tabellen nicht zu unterscheiden ob die 1:N-Beziehung optional ist oder nicht. Das hat für Sie die Auswirkung dass Sie eine 1:N-Beziehung immer so ableiten können, dass der Primärschlüssel der einen Tabelle Fremdschlüssel der anderen Tabelle wird.
Wir möchten es aber nicht versäumen darauf hinzuweisen, dass bei einer 1:N-Beziehung eine neue Tabelle entstehen kann, wenn die Beziehung optional ist. Der Grund hierfür ist, dass es eine Spalte für den Fremdschlüssel geben muss, diese aber nur selten mit Werten belegt würde (wenn die Merkmalsausprägung selten ist). Um dies zu verhindern, könnte man eine weitere Tabelle erzeugen. Diese Vorgehensweise ist in der Praxis allerdings eher selten. Daher verzichten wir auf eine weitere Diskussion.
Folgende Tabellen bilden die Beziehung ”Bürger besitzt Wohnungseigentum, Wohnungseigentum gehört Bürger“ ab:
Der Primärschlüssel von der Tabelle ”Bürger” wird Fremdschlüssel in der Tabelle ”Wohnungseigentum“. Die Beziehung kann nur so richtig aufgelöst werden. Den Primärschlüssel von der Tabelle ”Wohnungseigentum“ als Fremdschlüssel in die Tabelle ”Bürger” einzufügen würde dazu führen, dass man bei jedem weiteren Wohnungseigentum die Struktur der Tabelle Bürger ändern muss, da immer eine weitere Spalte hinzugefügt würde. Angenommen ein Bürger hätte 10 Wohnungen, ein anderer drei und wieder ein anderer hätte gar keine Wohnung, können Sie sich vorstellen wie zerklüftet die Tabelle wäre (Stichwort Schweizer Käse ;-).

Bei Datenbanken in Unternehmen gibt es Administratoren, welche für die Datenbank verantwortlich sind. Nur diese können dann die Struktur der Tabelle ändern. Die Benutzer können meistens nur Daten einfügen oder pflegen. Wenn Sie jeden Tag zig Mal den Datenbankadministrator wegen einer Strukturänderung der Tabelle belästigen, werden Sie sich damit keine Freunde machen.
Deswegen: Wann immer Sie auf einer der beiden Seiten als Kardinalität "1..1" stehen haben, können Sie den Primärschlüssel dieser Seite nehmen, und als Fremdschlüssel in der Tabelle der anderen Seite der Beziehung speichern. Es ist dabei völlig unwichtig, was als Kardinalität auf der anderen Seite steht.
Weiteres Beispiel:

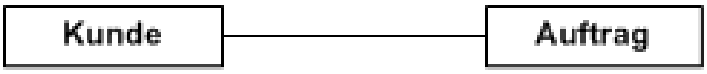


Abbildung 3.5
Beziehung Auftrag - Kunde

Auch hier wird der Primärschlüssel von Kunde der Fremdschlüssel in der Tabelle Auftrag.
Und noch ein Beispiel:

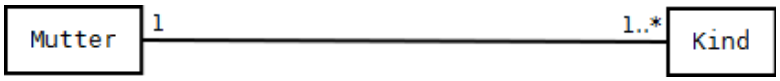


Abbildung 3.6
Mutter-Kind Muss-Beziehung

Auch hier wieder die gleiche Vorgehensweise. Der Primärschlüssel aus Mutter wird Fremdschlüssel in der Tabelle Kind.

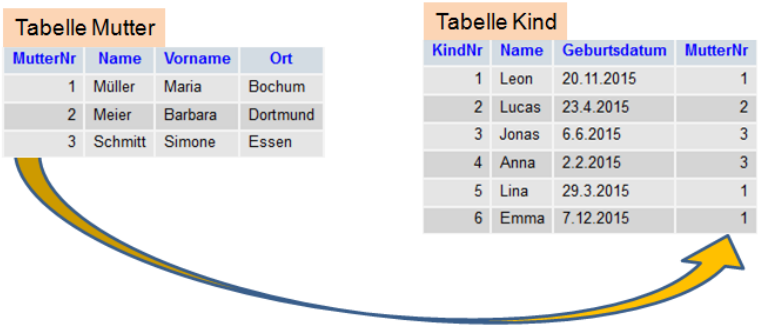


Abbildung 3.7
Mutter-Kind Beziehung Ableitung in Tabellen

3.3 m:n Beziehung

Die m:n ist vermutlich die leichteste Beziehung um davon die Tabellen abzuleiten. Bei dieser Beziehung entsteht immer eine neue Tabelle. Der Primärschlüssel der neu enstehenden Tabelle enthält mindestens die Primärschlüssel der Ausgangstabellen.
Ein Auftrag enthält mindestens einen bis beliebig viele Artikel.
Ein Artikel kann in einem Auftrag vorhanden sein.

Das ergibt folgende Tabellen:

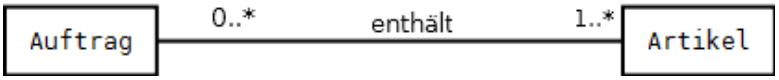


Abbildung 3.8
Auftrag - Artikel Beziehung

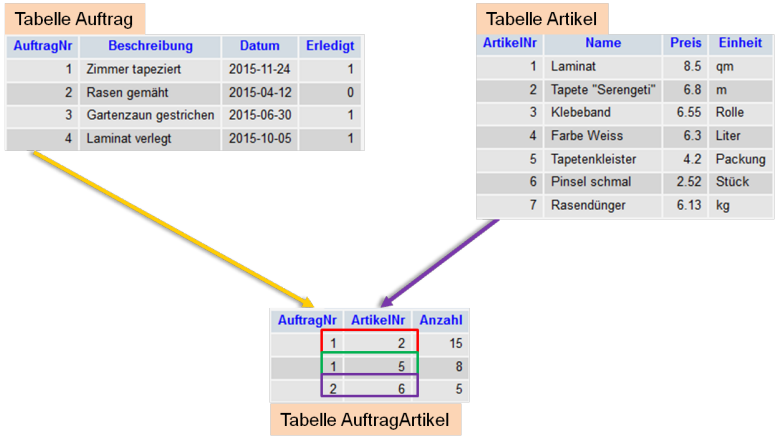


Abbildung 3.9
Auftrag - Artikel Ableitung

Es entsteht hier eine neue Tabelle, eine sogenannte Verbindungstabelle, mit dem Namen ”Auftrag Artikel”. Der Name ist zusammengesetzt aus den Ursprungstabellen. Man könnte die Tabelle auch “Rechnung“ oder “Auflistung“ oder sonst wie nennen. Wichtig ist, dass die Tabellenamen “sprechend“ sind, dass bedeutet sie sollen beschreiben welche Beziehungen bzw. Daten in ihnen vorhanden sind. Stellen Sie sich eine Datenbank mit 100 Tabellen vor. Die Tabellen heißen AAA, BBB, CCC, DDD usw. usw.. Wie lange müssen Sie suchen bis Sie die Tabelle mit den Kunden finden? Haben Sie aber die gleiche Datenbank mit dem Unterschied das die Tabellen treffende Namen haben, ist die Kundentabelle sehr schnell gefunden. Deswegen beim Tabellennamen immer treffende Bezeichnungen wählen.
Sie erkennen, dass die Primärschlüssel aus ”Auftrag” sowie ”Artikel” in die neue Tabelle wandern. In der neuen Tabelle ergeben diese beiden Fremdschlüssel jetzt einen zusammengesetzten Primärschlüssel. Verbindungstabellen besitzen immer einen zusammengesetzten Primärschlüssel. Eine einzelne Spalte kann nicht Primärschlüssel sein Betrachten Sie die Rechtecke, welche um die Zeilen der Tabelle AuftragArtikel gemalt wurden. Jedes Rechteck stellt einen Primärschlüssel dar. Ein Primärschlüssel muss eindeutig sein, und genau das ist dieser zusammengesetzte Primärschlüssel in diesem Fall. Die Verbindungstabelle besitzt ein weiteres Attribut. Dies muss nicht so sein, sondern ist hier nötig, weil die Stückzahl nicht in die Tabelle “Auftrag“ geschrieben werden kann und auch nicht in in die Tabelle ”Artikel”. Die Stückzahl ist vom Auftrag und vom Artikel abhängig und kann demnach nur in der Verbindungstabelle als zusätzliches Attribut stehen. Der zusammengesetzte Primärschlüssel muss nicht zwingend nur aus den Primärschlüsseln der Grundtabellen bestehen. Es kann durchaus sein, dass die Primärschlüssel der Grundtabellen allein nicht eindeutig sind. Dann müssen wir weitere Attribute zum Primärschlüssel hinzufügen.

3.4 Beziehungen Grad > 2

Hier erkennen wir drei Entitätstypen woraus folgt, dass hier mindestens drei Tabellen entstehen. Die Ableitung ist einfach: Es entsteht eine neue Tabelle. Der Primärschlüssel der neuen Tabelle besteht mindestens aus den Primärschlüsseln der Ursprungstabellen. Betrachten Sie bitte folgende Grafik 3.11:
Normalerweise hätten wir den Namen der neuen Tabelle aus den Namen der Ursprungstabellen zusammengesetzt. OnlineArtikelZeitraumWerbung ist aber nicht wirklich ein schöner Name, deswegen haben wir die neue Tabelle ”Veröffentlichung” genannt. Den Namen, welchen Sie bei der neu entstehenden Tabelle benutzen, können Sie frei vergeben. Achten Sie aber darauf, dass er sinnvoll ist.

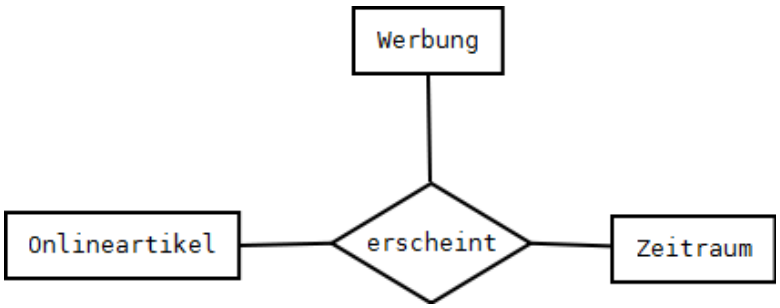


Abbildung 3.10
Beziehung Grad > 3

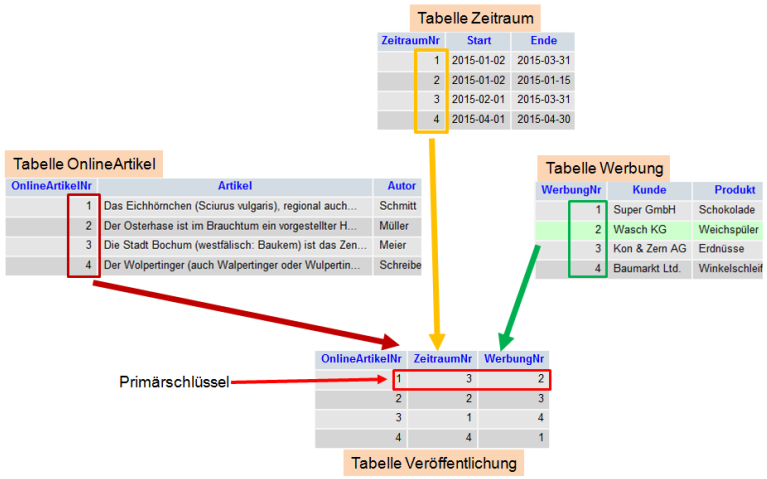


Abbildung 3.11
Beziehung Grad > 3 Ableitung

3.5 Generalisierung/Spezialisierung

Diese Beziehung wird in drei Tabellen abgebildet: Benutzer, Student sowie Professor. Der Primärschlüssel aus der Tabelle “Benutzer“ wird gleichzeitig Primärschlüssel in einer der beiden spezialisierten Tabellen. In die Tabelle “Benutzer“ kommen nur diejenigen Attribute, welche die beiden Benutzertypen gemeinsam haben, z.B. Anrede, Vorname, Nachname. In die jeweiligen speziellen Tabellen kommen diejenigen Attribute, welche speziell sind, wie z.B.: Die Matrikelnummer, ein Student hat eine, ein Professor nicht. Dafür hat ein Professor ein Fach in welchem er Vorlesungen hält und ein Büro, ein Student hat kein Büro[1], zumindest nicht an der Hochschule. In die Tabelle ”Student” wird demnach eine Spalte ”Matrikelnummer” angelegt, in die Tabelle “Professor“ wird eine Spalte “Büro“ eingefügt.

[1] Ok, es gibt auch die Spezies der studierenden Hochschulmitarbeiter mit Büro ;-)

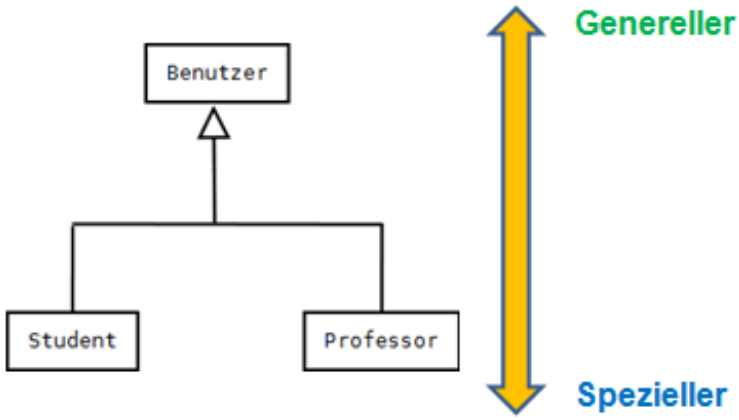


Abbildung 3.12
Generalisierung / Spezialisierung

Tabelle Benutzer

BenutzerNr	Vorname	Nachname	Ort
1	Otto	Mustermann	Bochum
2	Hans	Dampf	Dortmund
3	Hubert	Jäger	Essen
4	August	Lustig	Bochum
5	Gisela	Blumenfeld	Bochum
6	Peter	Michael	Bochum

Tabelle Student

BenutzerNr	MatrikelNummer	Semester	Studiengang
2	456456456	3	Bachelor of Something
3	987987987	2	Master of Something
6	123123123	5	Bachelor of ...

Tabelle Professor

BenutzerNr	Fach	B
1	Betriebswirtschaftslehre, insbesondere Betrieblich...	AW
4	Betriebswirtschaftslehre, insbesondere Wirtschafts...	AW
5	Betriebswirtschaftslehre	AW

Abbildung 3.13
Generalisierung / Spezialisierung Ableitung der Tabellen

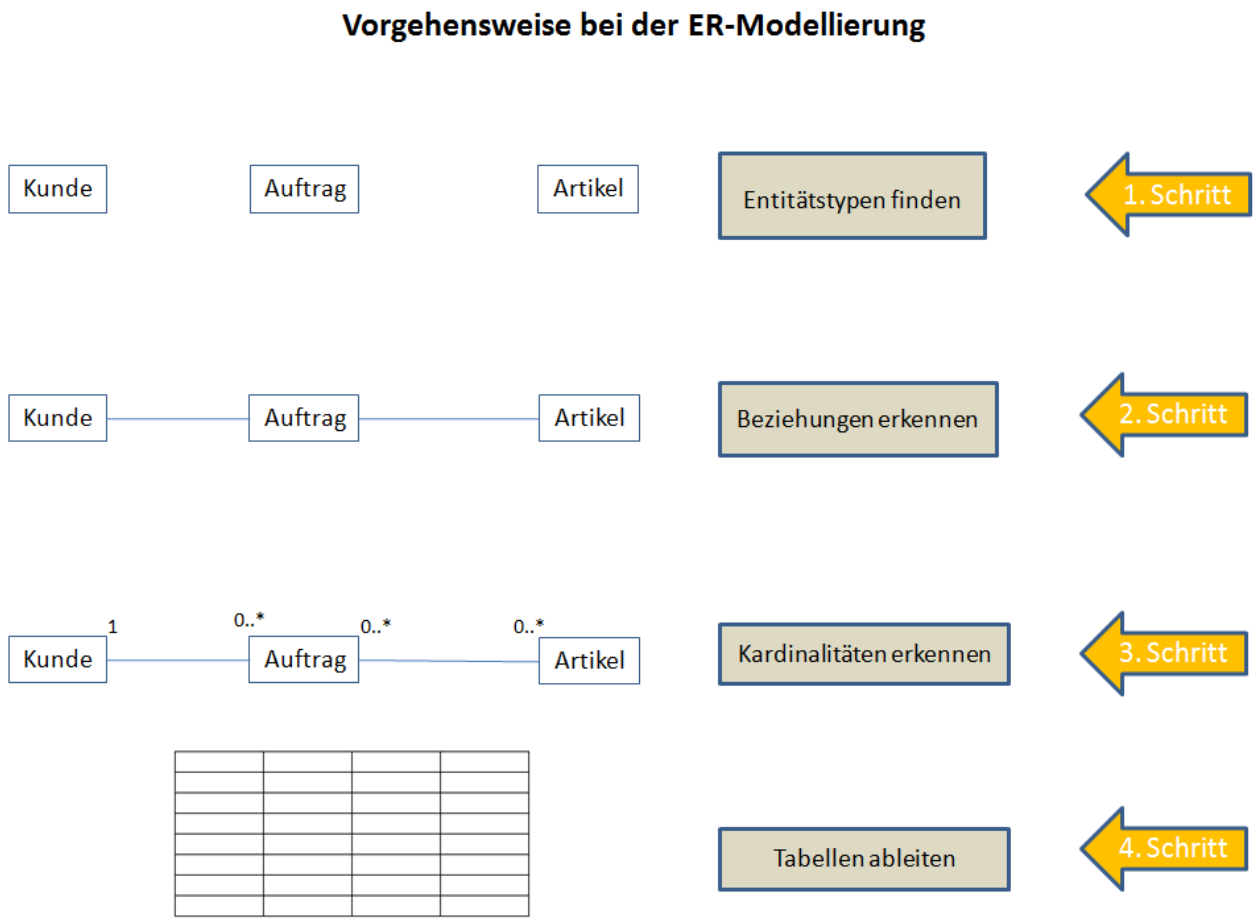


Abbildung 3.14
Komplette Vorgehensweise bei der ER-Modellierung

Kapitel 4

SQL

SQL bedeutet “Structured Query Language“. SQL ist ein Standard an den sich alle führenden Softwarehersteller orientieren. SQL ist sehr leicht erlernbar. Einzeln als Software können Sie SQL nicht kaufen, SQL ist in einem Datenbankmanagementsystem automatisch integriert. Als Datenbankmanagementsystem wird das bezeichnet, was Sie vermutlich unter dem Begriff Datenbank kennen, wie z.B. MySQL, Microsoft SQL-Server, Microsoft Access. Mittels SQL können Sie Datenbanken definieren, manipulieren und abfragen. Sie können alle Arbeiten welche in einer Datenbank notwendig sind mit SQL durchführen. Sie werden sich jetzt vielleicht fragen ”Warum soll ich SQL lernen wenn ich Abfragen mir mit einer Maus zusammenklicken kann”. Nun, sicherlich ist es möglich Abfragen mit einer Maus zusammenklicken und dafür sind keine Kenntnisse in SQL erforderlich. Wenn die Abfragen aber komplexer werden, wird es sehr schwierig, wenn nicht gar unmöglich, diese mit der Maus zu erstellen und spätestens dann benötigen Sie SQL. Natürlich benötigen Sie auch SQL-Kenntnisse, wenn Ihnen keine grafische Schnittstelle zur Verfügung steht.

Hier zeigen wir direkt ein erstes Beispiel:

```
Select Nachname, Vorname from Kunde
```

Abbildung 4.1
Einfacher SQL-Befehl

Der Befehl sagt aus, dass wir den Inhalt der Spalten “Nachname“ und “Vorname“ der Tabelle Kunde ausgeben möchten. Das wir die Daten aus der Tabelle Kunde möchten, ergibt sich aus “from Kunde“. “from“ ist ein SQL Schlüsselwort und dieses muss direkt nach den Spalten stehen. Kunde ist der Tabellenname. Würde die Tabelle jetzt nicht Kunde sondern ”Adressen” heißen würde die Abfrage lauten: “Select Nachname, Vorname from Adressen”. Bei dem Schreiben von SQL-Befehlen ist sorgfältiges Arbeiten gefragt. Ein Leerzeichen an der falschen Stelle oder statt einen Großbuchstaben ein Kleinbuchstabe kann zur Folge haben, dass Ihre Abfrage nicht funktioniert. Wobei es bei den SQL-Befehlen selbst irrelevant ist, ob diese groß oder kleingeschrieben werden (wird auch als “nicht case-sensitive“ bezeichnet). Anders sieht es bei den Spalten- sowie Tabellenamen aus. Hier ist es ggf. nicht egal ob Sie die Namen groß oder klein schreiben. Am besten schreiben Sie den Spalten- und den Tabellennamen so wie er auch in der Datenbank angezeigt wird.
Die nun folgende Grafik 4.2 zeigt Ihnen, mit welchen Schlüsselworten ein SQL-Befehl beginnen kann:

Es gibt natürlich noch viel mehr SQL-Befehle, aber da wir uns hier mit dem auswählen, ändern, löschen sowie einfügen von Daten beschäftigen, reichen uns die oben gezeigten vier Befehle aus. Mit ”Select” selektieren Sie Daten. Möchten Sie Daten löschen, verwenden Sie den Befehl ”Delete”. Möchten Sie bestehende Daten ändern benutzen Sie “Update“. Neue Daten werden mit ”Insert” eingefügt. Bei einer “Select“-Abfrage stehen nach dem Schlüsselwort “Select“ die Spalten welche Sie angezeigt bekommen möchten. Sollen alle Daten der Tabelle angezeigt werden benutzt man den ”*“:

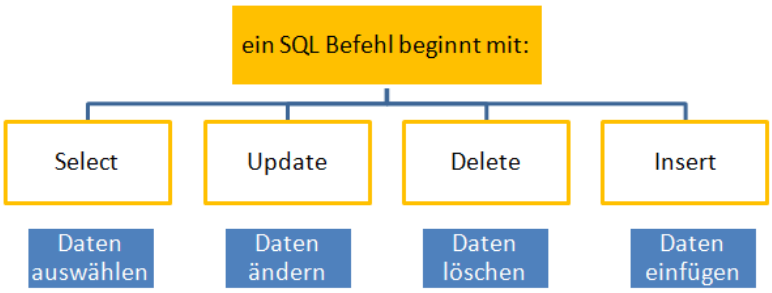


Abbildung 4.2
Übersichtsgrafik SQL Befehle

4.1 Datenselektion

4.1.1 SQL-1: Alle Daten einer Tabelle ausgeben

Aufgabe:
Lassen Sie sich alle Datensätze der Tabelle anzeigen.

SQL Statement

```
Select * from Kunde
```

Ergebnis

KundeNr	Vorname	Nachname	PLZ	Ort
1	Otto	Mustermann	44801	Bochum
2	Hans	Dampf	44135	Dortmund
3	Hubert	Jäger	45359	Essen
4	August	Lustig	44805	Bochum
5	Gisela	Blumenfeld	44801	Bochum
6	Peter	Michael	44795	Bochum
7	Agnes	Blumenfeld	44789	Bochum
8	Willi	Fragenzapf	45219	Essen

Abbildung 4.3
SQL-1: Alle Daten einer Tabelle ausgeben

Die einfachste aller Abfragen benutzt statt der Angabe der Spalten einen *. Der * bedeutet, dass der gesamte Inhalt der Tabelle, hier der Tabelle ”Kunde”, angezeigt wird. Die Ergebnisse werden nach dem Primärschlüssel aufsteigend sortiert. Dies ist bei MySQL die Standardsortierung. Diese Sortierung wird immer benutzt wenn Sie in der Abfrage keine Sortierung definieren.

4.1.2 SQL-2: Spalten auswählen

Aufgabe:
Lassen Sie sich alle Vor- und Nachnamen in der Tabelle Kunde ausgeben

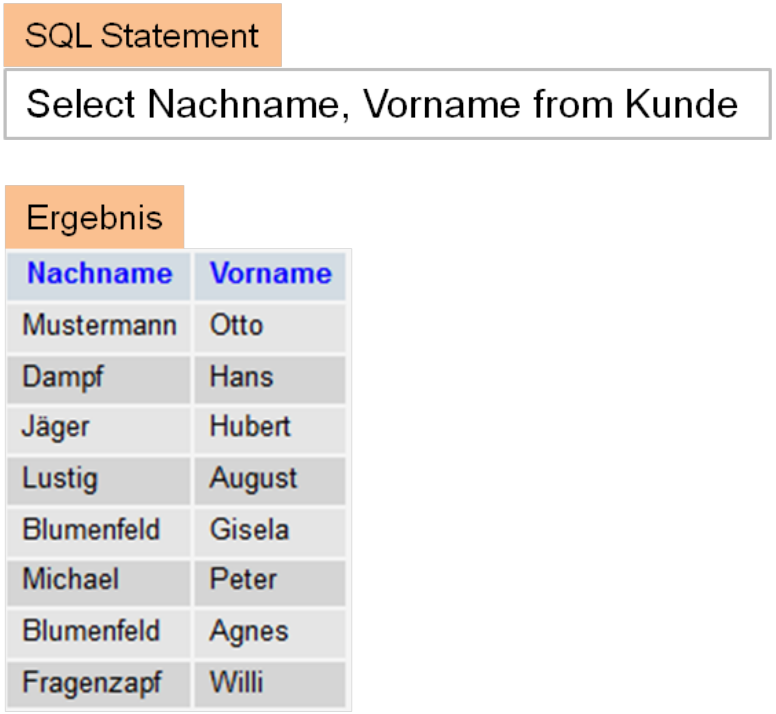


Abbildung 4.4
SQL 2: Bestimmte Spalten ausgeben

Die einzelnen Spalten werden durch Kommas getrennt. Nach dem Befehl “Select” muss ein Leerzeichen stehen, ebenso zwischen “from“ und “Kunde“. Kunde bezeichnet die Tabelle aus welcher Sie Daten holen möchten. Die Ausgabe ist unsortiert. Das ändern wir direkt mit der nächsten Aufgabe.

4.1.3 SQL-3: Sortierung aufsteigend

Aufgabe:
Lassen Sie sich alle Nach- und Vornamen aus der Tabelle Kunde aufsteigend sortiert ausgeben

Schon ist die Ausgabe nach den Nachnamen sortiert. Per Default wird immer aufsteigend sortiert. Das bedeutet: Geben Sie nicht explizit einen Sortiertyp an, erhalten Sie immer die aufsteigende Sortierung. Wir können die Liste aber auch ganz einfach reversiv sortieren lassen:

SQL Statement

```
Select Nachname, Vorname from Kunde order by Nachname
```

Ergebnis

Nachname	Vorname
Blumenfeld	Gisela
Blumenfeld	Agnes
Dampf	Hans
Fragenzapf	Willi
Jäger	Hubert
Lustig	August
Michael	Peter
Mustermann	Otto

Abbildung 4.5
SQL-3: Ausgabe sortieren lassen

4.1.4 SQL-4: Ergebnisse absteigend sortieren

Aufgabe:
Geben Sie Vor- und Nachname aus und sortieren Sie die Ergebnisliste absteigend nach Nachname

SQL Statement

```
Select Vorname, Nachname from Kunde order by Nachname desc
```

Ergebnis

Nachname	Vorname
Mustermann	Otto
Michael	Peter
Lustig	August
Jäger	Hubert
Fragenzapf	Willi
Dampf	Hans
Blumenfeld	Gisela
Blumenfeld	Agnes

Abbildung 4.6
SQL-4: Ausgabe absteigend nach einer Spalte sortieren lassen

Mit dem Befehl ”DESC” welches hinter Nachname steht, werden die Daten absteigend sortiert. ”ASC” ist der Befehl für die aufsteigende Sortierung, welche immer benutzt wird wenn nichts anderes angegeben wurde.

4.1.5 SQL-5: Distinct vermeidet doppelte Werte

Aufgabe:
Aus welchen Orten kommen Ihre Kunden? Vermeiden Sie doppelte Werte.

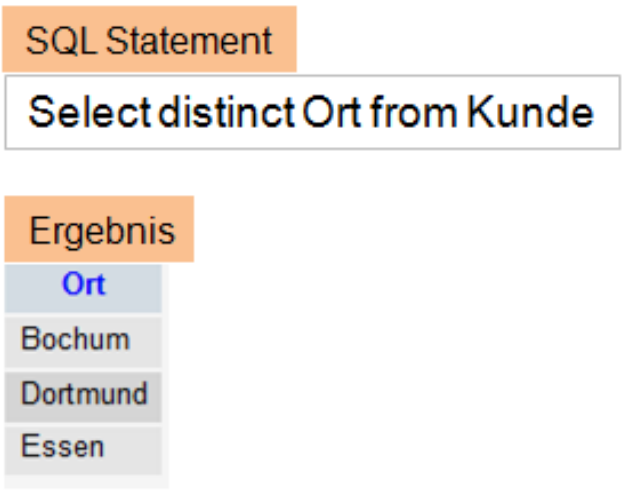


Abbildung 4.7
SQL-5: Doppelte Werte vermeiden

Durch Verwendung der Funktion “Distinct“ werden doppelte Werte einer Spalte eliminiert. Sie erkennen dass jeder Ort nur einmal ausgeben wird. Ohne die Funktion distinct würde z.B. der Ort Essen zweimal in der Liste erscheinen.

4.1.6 SQL-6: Sortierung nach mehreren Spalten

Aufgabe:
Geben Sie die Vor- und Nachnamen Ihrer Kunden aus Bochum aus. Sortieren Sie die Ausgabe absteigend nach Nachnamen. Sind Datensätze mit gleichen Nachnamen vorhanden, sortieren Sie die Datensätze absteigend nach Vorname.

SQL Statement

```
Select
Nachname, Vorname, Ort
from Kunde
where Ort="Bochum"
order by Nachname desc, Vorname desc
```

Ergebnis

Nachname	Vorname	Ort
Mustermann	Otto	Bochum
Michael	Peter	Bochum
Lustig	August	Bochum
Blumenfeld	Gisela	Bochum
Blumenfeld	Agnes	Bochum

Abbildung 4.8
SQL-6: Sortierung nach mehreren Spalten

Bei dieser Abfrage schränken wir die Ergebnismenge auf diejenigen Datensätze ein, welche in der Spalte “Ort“ den Wert “Bochum“ haben. Dazu benutzen wir den Befehl “where“, welcher direkt nach dem Tabellennamen steht. Hinter dem Befehl ”where” wird eine Bedingung definiert. Es ist selbstverständlich auch möglich mehrere Bedingungen zu definieren. Es werden nur die Datensätze selektiert, welche diese Bedingung(en) erfüllen. Die Bedingung bei dieser Abfrage lautet “where Ort = ’Bochum’“. Wenn die Bedingung Text enthält, und Text ist alles außer reinen Zahlen, muss der Wert in Anführungszeichen gesetzt werden. Bei dieser Abfrage ist neben der Bedingung ”Ort=’Bochum”’ die Sortierung zu beachten. Diese sortiert zunächst absteigend nach Nachname und anschließend nach Vornamen. Sind zwei gleiche Nachnamen vorhanden, wird anschließend absteigend nach Vornamen sortiert. Diese Sortierweise zeigt der Nachname “Blumenfeld”.
Hinter dem “WHERE“ kommt, wie bereits erklärt, eine Bedingung. Der Fachausdruck für diese Bedingung ist “boolescher Ausdruck“. Ein boolescher Ausdruck ist ein Ausdruck, der einen Wahrheitswert (wahr oder falsch) liefert. Im einfachsten Fall liefern Vergleiche solche Wahrheitswerte. Mögliche Vergleiche in SQL sind (siehe Tabelle 4.1):

IS NULL ist ein besonderer Ausdruck. Dieser prüft ob in einer Zelle nichts, also rein gar nichts steht. Haben Sie ein Leerzeichen in einer Zelle stehen wird diese durch ”IS NULL” nicht selektiert, weil ein Leerzeichen bereits einen Wert darstellt. Eine Beispiel mit IS NULL geben wir später.
Hier sehen Sie auch, dass Texte in einfachen Anführungszeichen (’: auf der Tastatur über dem #) oder in Anführungszeichen stehen müssen (Zahlen nicht). Wenn Sie nach Spalten sortieren, in denen Text enthalten ist, wird ähnlich wie im Telefonbuch alphabetisch sortiert.

4.1.7 SQL-7: Bedingungen (mehrere)

Aufgabe:
Prüfen Sie, ob eine Herr/Frau Blumenfeld im Postleitzahlgebiet 44801 wohnt.
Die Bedingungen einer Auswahl können auch logisch verknüpft werden. Dazu dienen die boolschen Operatoren (Tabelle 4.2:

Ausdruck	Bedeutung
<	größer als
>	kleiner als
=	ist gleich
>=	größer gleich
<=	kleiner gleich
<>	ungleich
!=	ungleich
≙	ungleich
IS NULL	Zellen ohne jeglichen Wert

Tabelle 4.1
Mögliche Vergleiche in SQL

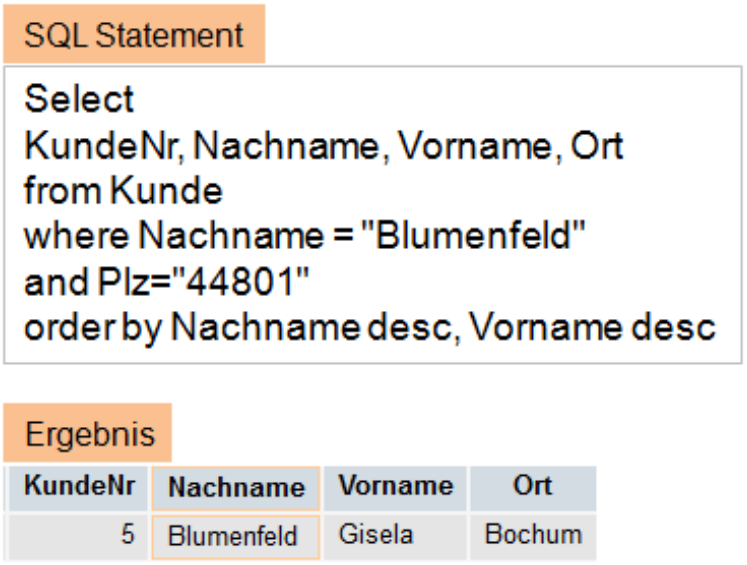


Abbildung 4.9
SQL-7: Abfrage mit mehreren Bedingungen

Wie bei der Grundrechenarten gibt es auch hier eine Vorrangsregel:
NOT wird vor AND und dies vor OR ausgeführt.
Für die Abfrage bedeutet dies: Im der Spalte Nachname muss der Wert Blumenfeld lauten und in der Spalte Plz muss der Wert 44801 lauten. Wenn beide Ausdrücke wahr sind wird der Datensatz angezeigt.

Ausdruck	Bedeutung
AND	(alle Bedingungen müssen erfüllt sein)
OR	(mindestens eine Bedingung muss erfüllt sein)
NOT	(die Bedingung darf nicht erfüllt sein)

Tabelle 4.2
Boolsche Operatoren

4.1.8 SQL-8: NOT Operator

Aufgabe:
Lassen Sie sich alle Kunden ausgeben welche nicht aus Bochum kommen

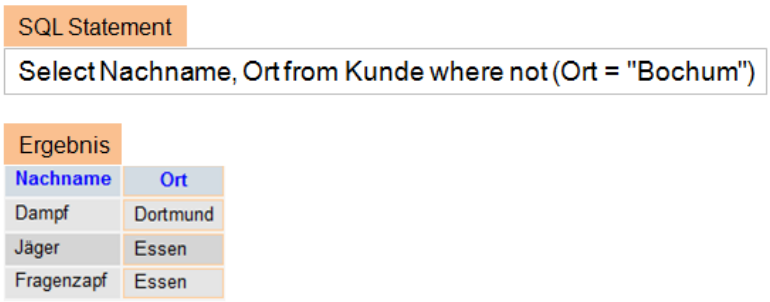


Abbildung 4.10
SQL-8: Verwenden des NOT-Operators

Durch die Bedingung 'not (Ort="Bochum")' werden natürlich die Kunden aus Bochum nicht angezeigt.

4.1.9 SQL-9: Setzen von Klammern

Aufgabe:
Geben Sie die Kunden aus welche dem Postleitzahlengebiet 44801 zugehören und Frederik oder Frederick heißen.

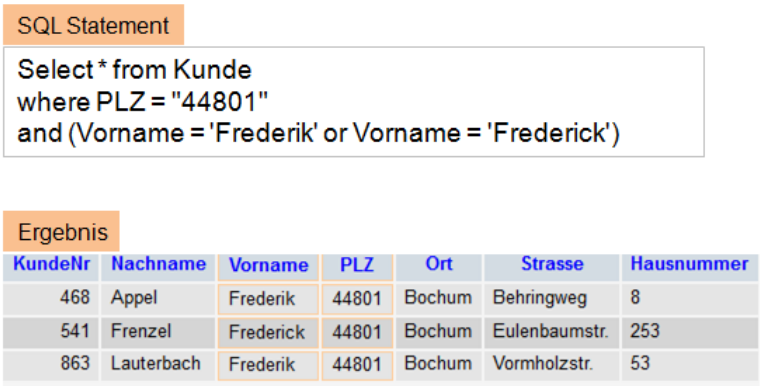


Abbildung 4.11
SQL-9: Klammersetzung

Durch das Setzen von Klammern werden die Vorrangregeln außer Kraft gesetzt. Lassen Sie die Klammern ganz weg, erhalten Sie folgendes Ergebnis:
Dies resultiert daraus, dass dann wieder "and" einen höheren Rang als "or" besitzt. Somit werden Datensätze selektiert, welche als Postleitzahl die 44801 haben und bei denen der Vorname "Frederik" oder "Frederick" lautet.

4.1.10 SQL-10: Verknüpfung von Ausdrücken

Aufgabe: Lassen Sie die Datensätze ausgeben bei denen entweder der Kunde die Kundennummer 1 hat und in Bochum wohnt oder alle Kunden aus Essen.

SQL Statement

```
Select Nachname from Kunde
where (KundeNr=1 and Ort='Bochum')
or Ort="Essen"
```

Ergebnis

Nachname
Mustermann
Jäger
Fragenzapf

Abbildung 4.12
SQL-10: Verknüpfung von Ausdrücken

Durch das Setzen von Klammern wird hier wieder das Ergebnis beeinflusst.

4.1.11 SQL-11: Rechnen mit SQL

Aufgabe:
Geben Sie für alle Artikel die Artikelnummer, die Bezeichnung, den Preis sowie den Bruttopreis aus.

SQL Statement

```
Select ArtikelNr, Bezeichnung, Preis, Preis *1.19 as Brutto from Artikel
```

Ergebnis

ArtikelNr	Bezeichnung	Preis	Brutto
1	Abschluss-Prüfungsaufgaben Fachoberschule /Berufso...	8.36	9.95
2	BWL kompakt: Grundwissen Betriebswirtschaftslehre	18.15	21.60
3	Statistische Methoden der VWL und BWL: Theorie und...	33.57	39.95
4	Grundwissen Wirtschaft: Wirtschaftsordnung - Unter...	8.36	9.95
5	BWL kompakt - die 100 wichtigsten Fakten	5.80	6.90
6	Training BWL /Rechnungswesen Realschule Bayern: Re...	8.36	9.95
7	Grundlagen der Betriebswirtschaftslehre: BWL	16.72	19.90
8	BWL mit Rechnungswesen für berufliche Gymnasien. A...	17.48	20.80
9	Mathematik für BWL-Bachelor: Schritt für Schritt m...	20.92	24.90

Abbildung 4.13
SQL-11: Rechnen mit SQL

Den Bruttopreis speichert man nie in der Tabelle, weil sich dieser aus dem Netto- oder Einkaufspreis + die Gewinnspanne + Umsatzsteuer berechnet. Da SQL rechnen kann, wird der Bruttopreis entweder direkt in der Abfrage berechnet oder in dem Programm das die Daten ausgibt. Der Bruttopreis, bzw. der Anteil der Umsatzsteuer, interessiert aber ein Unternehmen relativ wenig, da die Umsatzsteuer ein durchlaufender Posten ist. Dies ist noch ein Grund warum Bruttopreise nicht abgespeichert werden. Unternehmen kalkulieren nur mit Preisen ohne Umsatzsteuer.

4.1.12 SQL-12: SQL ohne Tabellen

An den Stellen, an denen bisher eine Spalte stand, darf auch ein Rechenausdruck stehen. Das bedeutet, dass die Datenbank für Sie Berechnungen durchführen kann. Wenn Sie z.B. bei den Preisen Nettopreise gespeichert haben, jetzt aber die Nettoverkaufspreise benötigen, können Sie diese einfach ausrechnen. Für jede Ausgabespalte kann durch ”AS” ein Alias vergeben werden, ein temporärer Name für diese Spalte. Insbesondere bei solchen Berechnungen wie sie in diesem Beispiel durchgeführt werden, kann so eine sinnvolle Ausgabe erzeugt werden. Wenn Sie den Alias weglassen, hängt es vom Datenbankmanagementsystem ab, welche Spaltenüberschrift angezeigt wird. Neben den Grundrechenarten gibt es noch eine Menge weiterer Funktionen, mit denen z.B. ein Datum manipuliert werden kann, Texte verändert werden oder mathematische Funktionen verwendet werden. Wenn Sie tatsächlich eine Berechnung durchführen sollen, dann schauen Sie in die Beschreibung Ihrer Datenbank, was diese an Funktionen zur Verfügung stellt. Übrigens kann SQL auch einfach so rechnen ohne dass Tabellen beteiligt sind:

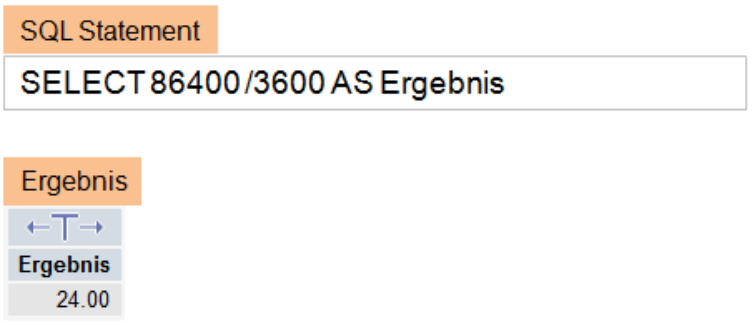


Abbildung 4.14
SQL-12: Rechnen ohne Tabellen

Das besondere hier ist, dass keine Tabelle im SQL Befehl angegeben wird und die Abfrage dennoch funktioniert. Dies ist aber nur dann der Fall, wenn wir keine Tabellenspalte angeben.

4.1.13 SQL-13: Funktionen in SQL

Aufgabe:
Zeigen Sie die Anzahl der Datensätze, den maximalen Preis, den kleinsten Preis, den Durchschnittspreis und die Summe aller Preise in der Tabelle Artikel.
Um diese Aufgabe zu lösen benötigen wir Funktionen. Funktionen sind kleine Programme welche von SQL zur Verfügung gestellt werden. Es gibt sehr viele Funktionen in SQL, daher zeigen wir hier nur eine paar davon. Sicherlich wäre es möglich den maximalen Preis durch “Select Preis from Artikel order by Preis desc“ zu selektieren, mit der Funktion “max(Preis) geht dies aber schneller und eleganter. Auch ist es mittels Funktionen möglich verschiedenste Abfragen in einem SQL-Statement zusammenzufassen. Wie das geht sehen Sie oben in der Abfrage.

4.1.14 SQL-14: Suche nach Text mit Platzhaltern

Aufgabe: Finden Sie alle Kunden deren Nachname mit dem Buchstaben “M“ beginnt und mit “er“ aufhört.
Bei dieser Abfrage benutzen wir den Ausdruck “like ’M%er“. Mit diesem Ausdruck erreichen wir, das alle Nachnamen gefunden werden welche mit “M“ beginnen und mit “er“ aufhören. Zwischen M und er kann alles Mögliche stehen, wie z.B.: M123456er oder MeineErsteGeburtstagsfeier usw. Das “Kunden gefunden deren Nachname auf “mann“ endet, wie z.B. Baumann, Bergmann, Hofmann usw. Analog verhält es sich bei “like ’Bauwelche mit “Bau“ beginnen, unabhängig

SQL Statement

```
Select
count(*) As "Anzahl Datensätze",
max(Preis) As "Maximaler Preis",
min(Preis) As "Minimaler Preis",
avg(Preis) As Durchschnittspreis,
sum(Preis) As "Summe der Preise"
from Artikel
```

Ergebnis

Anzahl Datensätze	Maximaler Preis	Minimaler Preis	Durchschnittspreis	Summe der Preise
7	8.5	2.52	5.86	41

Abbildung 4.15
SQL-13: Verwendung von Funktionen in SQL

SQL Statement

```
Select Nachname from Kunde where Nachname like "M%er"
```

Ergebnis

Nachname
Müller
Meyer
Meier
Maier
Mayer
Möller
Müller
Meyer
Meier

Abbildung 4.16
SQL-14: Suche nach Text mit Platzhaltern

davon was nach “Bau“ kommt, wie z.B.: Baumann, Baumeister usw.. Jokerzeichen bei LIKE:
% repräsentiert eine beliebige Folge von Zeichen.
_ repräsentiert ein einzelnes beliebiges Zeichen

4.1.15 SQL-15: Suchfunktionen

Aufgabe:
Finden Sie die Kunden deren Nachnamen aus fünf Buchstaben besteht. Der Nachname fängt mit “Me“ an und hört mit “er“ auf. Sinn und Zweck dieser Abfrage könnte sein, dass Sie nicht genau wissen wie Meier geschrieben wird. Deswegen lassen Sie sich alle Schreibweisen von Meier anzeigen (zumindest diejenigen welche mit ”ME” beginnen und mit “ER“ aufhören).

4.1.16 SQL-16: Selektion von NULL-Werten

Aufgabe: Finden Sie Artikel ohne Preis.
Wenn Sie in die Datenbank schauen, können Sie feststellen, dass bei einigen Artikeln kein Preis angegeben ist - hier steht in der Spalte Preis der Wert ”Null”. Damit ist gemeint, dass hier nichts eingetragen ist, dieses Attribut also bei diesem

SQL Statement

```
Select Nachname from Kunde where Nachname like "Me_er
```

Ergebnis

Nachname
Meyer
Meier
Meyer
Meier
Meyer
Meier

Abbildung 4.17
SQL-15: Jokerzeichen bei LIKE

SQL Statement

```
Select * from Artikel where Preis IS NULL
```

Ergebnis

ArtikelNr	Bezeichnung	Preis
25	BWL Grundwissen	*NULL*
38	Statistische Methoden der VWL und BWL. Theorie und...	*NULL*
43	Eine Einführung in die VWL, Bd.1, Markt und Wettbe...	*NULL*
44	Formelsammlung der Wirtschaftswissenschaften: Schw...	*NULL*
45	Formelsammlung der Wirtschaftswissenschaften: Schw...	*NULL*

Abbildung 4.18
SQL-16: NULL-Werte auswählen

Artikel leer ist. Hier können wir uns die Sinnhaftigkeit des Wertes ”Null” in Tabellen veranschaulichen. Stellen wir uns vor, dass wir Artikel einfügen wollen, für die der Preis noch nicht fest liegt. Hier können wir trivialerweise noch keinen Preis eintragen. Welchen Wert sollen wir stattdessen eintragen? Den Leerstring können wir nicht eintragen, denn der Preis ist ein Zahlfeld und der Leerstring ist keine Zahl. Jede Zahl ist aber ein gültiger Preis. Für diese Fälle ist der Wert ”Null” vorgesehen. Wie finden wir jetzt alle Artikel ohne Preis? Daher bietet SQL diesen speziellen Ausdruck “IS NULL“ an, mit dem abgefragt werden kann, ob ein Wert ”null” (also nicht vorhanden) ist.

4.1.17 SQL-17: Datensätze zählen

Aufgabe:
Zählen Sie:

- Alle Datensätze in der Tabelle Artikel
- Diejenigen Artikel welche einen Preis haben
- Wieviel unterschiedliche Preise in der Tabelle Artikel vorhanden sind

Auf der vorhergehenden Seite haben Sie gesehen, dass Sie mit count(*) die Anzahl von Datensätzen zählen können. Es werden hierbei alle gefundenen Datensätze gezählt. Geben Sie anstelle des Stern einen Spaltenname an, so wird gezählt, bei wie vielen Datensätzen in der angegeben Spalte ein Wert vorhanden ist. Sie sehen hier an der Differenz zwischen count(*) und count(Preis), dass 55 Artikel scheinbar gar keinen Preis besitzen. Mit Hilfe von count(distinct Spaltenname) können Sie auch zählen, wie viele unterschiedliche Werte in einer Spalte es gibt. Es gibt also zwar 364

SQL Statement

```
Select count(*) As 'Alle Datensätze', count(Preis) As 'Datensätze mit
Preis', count(distinct Preis) As 'Unterschiedliche Preise' from Artikel
```

Ergebnis

Alle Datensätze	Datensätze mit Preis	Unterschiedliche Preise
421	366	125

Abbildung 4.19
SQL-17: Datensätze zählen

Artikel mit Preisen, es gibt aber nur 125 unterschiedliche Preise. Die Suche könnte hier auch mit einem where-Statement eingeschränkt werden, es könnten also z.B. auch die Anzahl aller Preise gesucht werden, die höher als 20 €sind:
Select count(*) As “Alle Datensätze“, count(Preis) As “Datensätze mit Preis“, count(distinct Preis) As “Unterschiedliche Preise” from Artikel where Preis > 20
Da die Bezeichnungen für die Datenausgabe ein Leerzeichen beinhalten, wie z.B. bei “Alle Datensätze“, muss die Bezeichnung in der Abfrage in Hochkommatas gesetzt werden.

4.1.18 SQL-18: Gruppierung

Aufgabe:
Geben Sie die Anzahl der Kunden in den verschiedenen Städten aus. Benutzen Sie hierfür nur eine Abfrage.

SQL Statement

```
Select Ort, count(*) as Anzahl from Kunde Group by Ort
```

Ergebnis

Ort	Anzahl
Bochum	5
Dortmund	1
Essen	2

Tabelle

KundeNr	Vorname	Nachname	PLZ	Ort
1	Otto	Mustermann	44801	Bochum
2	Hans	Dampf	44135	Dortmund
3	Hubert	Jäger	45359	Essen
4	August	Lustig	44805	Bochum
5	Gisela	Blumenfeld	44801	Bochum
6	Peter	Michael	44795	Bochum
7	Agnes	Blumenfeld	44789	Bochum
8	Willi	Fragenzapf	45219	Essen

Abbildung 4.20
SQL-18: Gruppierung in SQL

Wir haben in der ursprünglichen Tabelle fünf Datensätze mit Kunden aus Bochum (Wert des Attributs Ort ist Bochum), zwei Datensätze mit Kunden aus Essen (Wert des Attributs Ort ist Essen) und einen Datensatz mit einem Kunden aus Dortmund (Wert des Attributs Ort ist Dortmund). Wird nach einem Attribut gruppiert (SQL-Anweisung “Group By“), so ermittelt das Datenbanksystem die unterschiedlichen Ausprägungen des Attributs nach denen gruppiert wird. In unserem Beispiel wird nach Ort gruppiert. Die unterschiedlichen Ausprägungen sind Bochum, Essen und Dortmund. Dann werden die Datensätze, wo die Ausprägungen des Gruppierungsattributs übereinstimmen, zu einem Datensatz zusammengeführt. Die der Gruppierung zugrundeliegende Vorgehensweise hat natürlich auch Auswirkungen auf die Attribute, die im Select-Teil einer SQL-Gruppierungsabfrage erlaubt oder sinnvoll sind.

4.1.19 SQL-19: Gruppierung Fehler

Aufgabe: Gruppieren Sie falsch ;-)

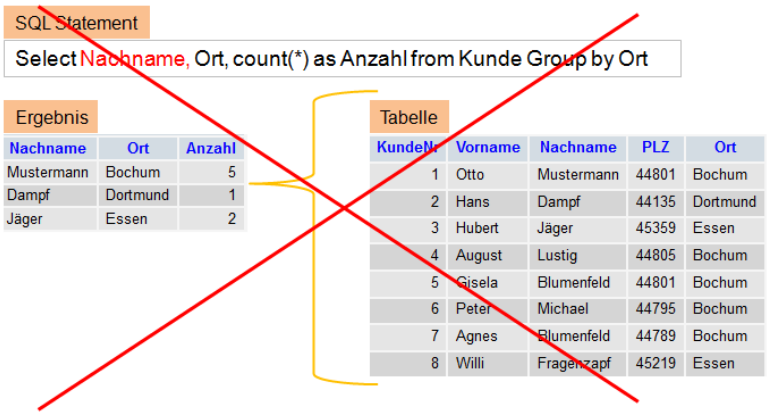


Abbildung 4.21
SQL-19: Fehler bei der Gruppierung

Hier soll der Name der Kunden zusätzlich im Ergebnis ausgewiesen werden. In unserer Ergebnismenge gibt es aber pro Stadt nur einen Datensatz. Wir haben aber mehr als einen Kunden pro Ort. Wie kann das Datenbanksystem aber mehrere unterschiedliche Namen in einem Datensatz darstellen? Die Antwort ist: Gar nicht. Das Datenbanksystem würfelt und stellt einen zufällig ausgewählten Namen dar. Ein Attribut mit unterschiedlichen Attributsausprägungen in den zusammenzufassenden Datensätzen macht also im Select-Teil wenig Sinn. Aufgenommen werden können:

- Das Attribut, nachdem gruppiert wird (in unserem Beispiel Ort).
- Die in 4.16SQL-14 gezeigten Funktionen.

So würde z.B. die Funktion max(KundeNr) im Select-Teil Sinn machen, denn das Datenbanksystem kann in den Ergebnisdatensätzen die größte KundeNr darstellen (in unserem Beispiel 7 für Bochum und 8 für Essen).

4.1.20 SQL-20: Having-Einschränkung bei Gruppierungen

Aufgabe:
Welche Artikel haben eine durchschnittliche Bewertung von mindestens 4?
Mit
Select avg(Wert) As Bewertung from Bewertung
kann der Durchschnittswert für alle Artikel berechnet werden. Mit
Select ArtikelNr, avg(Wert) As Bewertung from Bewertung group by ArtikelNr
wird der Durchschnittswert für jeden einzelnen Artikel berechnet (durch die Gruppierung). Dies ist auf der Folie links dargestellt. Mit
Select ArtikelNr, avg(Wert) As Bewertung from Bewertung
where Bewertung >= 4
group by ArtikelNr
wird das oben genannte Ziel nicht erreicht. Es werden bei der Berechnung des Durchschnitts nur die Bewertungen verwendet, die mindestens 4 sind. Schlechte Bewertungen fließen in die Berechnung gar nicht erst ein. Wir wollen aber erst den Durchschnitt für alle Bewertungen (je Artikel) berechnen, und danach diejenigen ausschließen, bei denen der Durchschnitt kleiner als 4 war. Der “Where“-Teil eines SQL-Statements bezieht sich auf die Daten vor der Gruppierung (also welche Daten werden zur Gruppierung herangezogen). Wenn Sie nur bestimmte Gruppen anzeigen wollen, z.B. nur die Gruppen, die mehr als 10 Elemente haben, weil sonst eine statistische Auswertung keinen Sinn macht, können Sie erst nach der Gruppierung eine Auswahl machen (vorher kennen Sie die Gruppen ja noch gar nicht). Um unter den Teilmengen/Gruppen, die sich ergeben haben, eine Auswahl zu machen, gibt es den ”having”- Teil eines Statements. Die Einschränkungen, die man mit ”having” macht, beziehen sich immer auf eine Gruppe (also auf das Ergebnis nach der Gruppierung).

SQL Statement

```
Select ArtikelNr, avg(Wert) as Bewertung from Bewertung
group by ArtikelNr
having avg(Wert) >=4
```

Ergebnis

ArtikelNr	Bewertung
1	4.6667
2	4.0000
4	5.0000
7	4.8000
9	4.5000
10	5.0000
13	5.0000
14	5.0000
15	5.0000

Abbildung 4.22
SQL-20: Einschränkungen von Gruppierungen durch HAVING

4.1.21 SQL-21: Abfrage über zwei Tabellen

Aufgabe:
Die Meier AG beschwerte sich das zwischen der Bestellung und der Lieferung zu viel Zeit vergehen würde. Überprüfen Sie ob das stimmt. Lassen Sie sich den Kundennamen, die Postleitzahl, den Ort sowie das Auftrags- und das Lieferdatum anzeigen

SQL Statement

```
Select
Kunde.Name, Kunde.PLZ, Kunde.Stadt, Auftrag.Auftragdatum, Auftrag.Lieferdatum
from Kunde, Auftrag
where Kunde.Name = "Meier AG"
and Kunde.KundeNr = Auftrag.KundeNr
```

Ergebnis

Name	PLZ	Stadt	Auftragdatum	Lieferdatum
Meier AG	47543	Bochum	2011-10-08	2011-10-09
Meier AG	47543	Bochum	2011-12-09	2011-12-16
Meier AG	47543	Bochum	2011-12-09	2011-12-30
Meier AG	47543	Bochum	2011-12-09	2011-12-29

Abbildung 4.23
SQL-21: Abfrage über zwei Tabellen

Da wir die Informationen, die wir in der Datenbank abspeichern, auf mehrere Tabellen verteilen, besteht in der Ausgabe die Notwendigkeit, die über mehrere Tabellen verteilten Informationen wieder insgesamt darzustellen. Dazu müssen wir Abfragen über mehrere Tabellen formulieren. SQL bietet natürlich diese Möglichkeit. Eine solche Abfrage wird als Join bezeichnet. Am Grundgerüst der Select-Abfragen ändert sich nichts. Betrachten wir zunächst die Tabellen Kunde und Auftrag. Die Information, dass ein Auftrag einem Kunden zugeordnet ist, wird über den Fremdschlüssel KundeNr in der Tabelle Auftrag gespeichert. Ein Kundendatensatz und ein Bestellungsdatensatz gehören immer dann zusammen, wenn die Kundennummer, die im Kunden gespeichert ist, gleich der Kundennummer ist, die in Auftrag gespeichert wurde.
In der Tabelle Kunde findet man die Kundennummer der Meier AG. Die Kundennummer benötigt man um in der Tabelle “Auftrag“ die einzelnen Aufträge der Meier AG zu finden. Die Meier AG hat die Kundennummer 2. Die Kundennummer ist die Verbindung zwischen der Tabelle “Kunde “ sowie der Tabelle “Auftrag“. Wir suchen alle Aufträge der Kunden-

nummer 2. Daraus entsteht folgende Abfrage:

- Im “Select-Teil“ schreiben wir den Tabellennamen vor die Namen der Felder. Dies ist ziemlich einleuchtend. Wir haben jetzt eine Abfrage über mehrere Tabellen. Innerhalb einer Tabelle müssen die Namen der Felder natürlich eindeutig sein (Wir können nicht zwei Felder gleichen Namens in einer Tabelle haben). Bei mehreren Tabellen ist das nicht mehr so. In unserem Beispiel gibt es ein Attribut KundeNr z.B. sowohl in der Kunde-, als auch in der Auftrags-Tabelle. Der Tabellenname zusammen mit dem Feldnamen hingegen ist wieder eindeutig. Der Feldname wird über einen Punkt (.) an den Tabellennamen angeschlossen.
- Im “From-Teil“ stehen alle Tabellen, die zur Erzeugung des Ergebnisses notwendig sind. In unserem Beispiel sind das die Tabellen Kunde und Auftrag. Beachten Sie, dass dies auch notwendig ist, wenn eine Tabelle nicht in der Ausgabe vorkommt.
- Der “Bedingungsteil“ wird um die Fremdschlüssel-Primärschlüssel-Beziehung ergänzt. In Worten heißt die im Bedingungsteil formulierte Anweisung an das Datenbanksystem:

1. Ermittle den (oder die) Datensatz (Datensätze) in der Tabelle Kunde, wo das Feld Name den Wert ”Meier AG“ hat.
2. Ermittle in diesem Datensatz die Kundennummer (in unserem Beispiel die Kundennummer 2).
3. Gehe mit dieser Kundennummer in die Tabelle Auftrag und ermittle die Datensätze in Auftrag, bei denen die Kundennummer (der Fremdschlüssel) mit der ermittelten Kundennummer in Kunde übereinstimmt.
4. Gebe die zugehörigen Datensätze aus.

4.1.22 SQL-22: Abfrage über zwei Tabellen

Aufgabe:
Geben Sie die Auftrags- sowie die Lieferdaten der Meier AG aus.

SQL Statement

```
Select Auftrag.Auftragdatum, Auftrag.Lieferdatum
from Kunde, Auftrag
where Kunde.Name='Meier AG'
and Kunde.KundeNr=Auftrag.KundeNr
```

Ergebnis

Auftragdatum	Lieferdatum
2011-10-08	2011-10-09
2011-12-09	2011-12-16
2011-12-09	2011-12-30
2011-12-09	2011-12-29

Abbildung 4.24
SQL-22: Abfrage über zwei Tabellen, aber nur Werte von einer Tabelle werden ausgegeben

Im dem vorherigen Beispiel wurden Spalten aus der Tabelle “Kunde“ sowie Spalten aus der Tabelle “Auftrag“ gleichzeitig im Ergebnis angezeigt. In dem jetzigen Beispiel lassen wir nur Daten aus der Tabelle “Auftrag“ anzeigen. Das Besondere hierbei ist, dass wir dennoch die Tabelle Kunde mit in die Abfrage aufnehmen müssen, weil wir nur die Daten von der Meier AG sehen möchten. Der Name ’Meier AG’ steht aber in der Tabelle Kunde, und somit ist es wieder erforderlich beide Tabellennamen in der Abfrage anzugeben:

4.1.23 SQL-23: Abfrage über zwei Tabellen

Aufgabe:
Suchen Sie alle Lieferungen des Monats Dezember im Jahr 2015.

SQL Statement

```
Select Kunde.Name, Kunde.PLZ, Kunde.Stadt, Auftrag.Lieferdatum
from Kunde, Auftrag
where YEAR(Auftrag.Lieferdatum) =2015
and MONTH(Auftrag.Lieferdatum) =12
and Kunde.KundeNr=Auftrag.KundeNr
order by Auftrag.Lieferdatum
```

Ergebnis

Name	PLZ	Stadt	Lieferdatum
Meier AG	47543	Bochum	2015-12-16
Schulz GmbH	44701	Bochum	2015-12-17
Schulz GmbH	44701	Bochum	2015-12-20
Schulz GmbH	44701	Bochum	2015-12-22
Fischer	44787	Bochum	2015-12-24
Schulz GmbH	44701	Bochum	2015-12-27
Meier AG	47543	Bochum	2015-12-29
Meier AG	47543	Bochum	2015-12-30

Abbildung 4.25
SQL-23: Abfrage über zwei Tabellen, Einschränkung durch Bedingungen

Hier müssen wir wieder aus zwei Tabellen unsere Daten sammeln. Der Name, Plz und die Stadt des Kunden stehen in der Tabelle Kunde und das Lieferdatum steht in der Tabelle Auftrag. Damit wir das Jahr 2015 und den Monat Dezember selektieren können, benutzen wir zwei Funktionen: YEAR und MONTH. Diese beiden Funktionen extrahieren aus einem Datum das Jahr bzw. den Monat (den Tag ggf. natürlich auch, das wäre dann die Funktion DAY). Dieser Wert kann dann verglichen werden, wie aus der Abfrage ersichtlich. Die Abfrage macht folgendes:

1. Sie ermittelt die Datensätze in der Tabelle Auftrag, bei denen das Feld Lieferdatum den Wert 2015 sowie den Wert 12 beim Monat enthält.
2. Anschließend wird von diesen Datensätzen die Kundennummer ermittelt.
3. Der Abgleich der Kundennummer erfolgt dann mit der Tabelle 'Kunde'. Es wird in der Tabelle 'Kunde' derjenige Datensatz ermittelt welcher die Kundennummer 2 hat.
4. Danach werden die Informationen ausgegeben

Wir zeigen Ihnen nun eine andere Möglichkeit, Abfragen über zwei oder mehr Tabellen durchzuführen. Die Ergebnisse sind natürlich gleich, die Schreibweise ist lediglich eine andere.

4.1.24 SQL-24: Joins

Aufgabe:
Geben Sie den Kundennamen sowie das Lieferdatum aus. Benutzen Sie left sowie right join.
Wenn Sie die Grafik betrachten fällt Ihnen auf, dass bei der rechten Abfrage die Tabellen "vertauscht" sind. Bei der Notationsweise mit dem Wort "join" ist es irrelevant welche Tabelle zuerst genannt wird. Bislang haben Sie gelernt, wie Informationen aus zwei Tabellen vermittels SQL-Anweisungen zusammengeführt werden können. Ist die Information auf mehr als zwei Tabellen verteilt, was z.B. bei n-m-Beziehungen der Fall ist, ändert sich die Vorgehensweise aber nicht. Wir formulieren im "Select-Teil" welche Felder wir im Ergebnis sehen wollen. Im "From-Teil" führen wir alle Tabellen auf, die zur Ermittlung des Ergebnisses benötigt werden. In den "Bedingungsteil" werden die Einschränkungen und über "and"

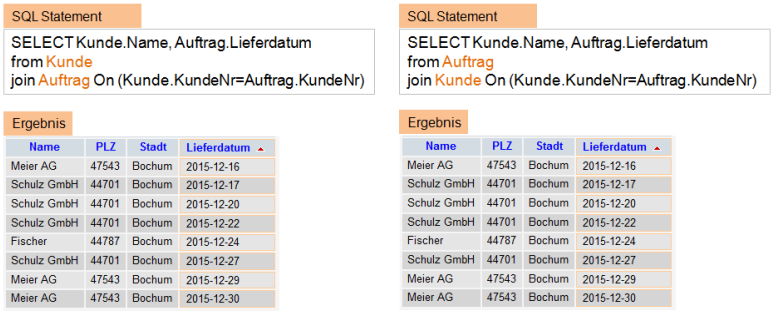


Abbildung 4.26
SQL-24: Joins in SQL

verbunden die Primärschlüssel - Fremdschlüsselbeziehungen aufgenommen. Wir zeigen dies an folgendem Beispiel: Wir möchten die Lieferdaten und die AuftragNr aller Aufträge ausgeben, in denen das Produkt ”USB-Stick (1GB)” enthalten ist. Aus unserer Tabellenstruktur wissen wir:

- Der Name der Produkte ist Attribut in Produkt
- Das Lieferdatum ist Attribut in Auftrag
- Diese beiden Tabellen sind über die Tabelle auftragProdukt miteinander verbunden

Die Lösung lautet:

4.1.25 SQL-25: Abfrage über drei Tabellen

Aufgabe:
Geben Sie die Auftragsnummer sowie das Lieferdatum aus.
Da die darzustellende Information jetzt auf drei Tabellen verteilt ist, benötigen wir im “From-Teil“ nun die drei in die Abfrage involvierten Tabellen: Auftrag, Produkt und auftragProdukt. Die Reihenfolge im “From-Teil“ ist irrelevant. Wichtig ist nur dass diese drei Tabellen aufgeführt sind. Im Bedingungsteil werden nun zwei Zeilen zur Abbildung der Primärschlüssel - Fremdschlüsselbeziehungen benötigt. In Worten heißt die im “Bedingungsteil“ formulierte Anweisung an das Datenbanksystem:

1. Ermittle den (oder die) Datensätze der Tabelle Produkt, bei denen der Wert des Feldes Name “USB-Stick (1GB)“ ist.
2. Ermittle in der Tabelle Produkt die zugehörige ProduktNr (2 ist das Ergebnis).
3. Ermittle in der Tabelle auftragProdukt die Datensätze, wo der Wert des Feldes ProduktNr der in Schritt 2 gefundenen entspricht.
4. Ermittle in der Tabelle auftragProdukt die zugehörigen AuftragNr”s.
5. Ermittle in der Tabelle Auftrag die Datensätze, wo der Wert des Feldes AuftragNr der in Schritt 4 gefundenen entspricht.
6. Ermittle das zugehörige Lieferdatum.
7. Gib die ermittelten Informationen aus.

Abschließend wollen wir noch die Namen der Kunden, in deren Aufträgen das Produkt “USB-Stick (1GB)“ enthalten ist, ausgeben. Dies ist eine einfache Erweiterung der Abfrage:

SQL Statement

```
Select Auftrag.AuftragNr, Auftrag.Lieferdatum
from Auftrag, auftragProdukt, Produkt
where Produkt.Name="USB-Stick (1GB)"
and Produkt.ProduktNr=auftragProdukt.ProduktNr
and auftragProdukt.AuftragNr=Auftrag.AuftragNr
```

Ergebnis

AuftragNr	Lieferdatum
1	2015-10-10
2	2015-10-09
7	2015-12-20
8	2015-12-27
9	2015-12-30
11	2015-12-24
13	2015-12-29

Abbildung 4.27
SQL-25: Abfrage über drei Tabellen

4.1.26 SQL-26: Abfrage über vier Tabellen

Aufgabe:
Geben Sie die den Kundennamen, die Auftragsnummer sowie das Lieferdatum aus.

4.1.27 SQL-27: join

Aufgabe:
Suchen Sie Kunden mit offenen Aufträgen.
Wir möchten alle Kunden anzeigen lassen, unabhängig davon ob sie einen Auftrag offen haben oder nicht. Bei den Kunden, zu denen es Aufträge gibt, soll zusätzlich die AuftragNr der Aufträge dargestellt werden.
Mit dem was Sie bisher gelernt haben, geht das nicht. Denn wenn wir die Primärschlüssel - Fremdschlüsselbeziehung in die Abfrage aufnehmen, werden nur solche Datensätze angezeigt, wo die KundeNr sowohl in Auftrag, als auch in Kunde vorkommt. Dies bedeutet, Kunden ohne Aufträge werden nicht angezeigt. Nehmen wir hingegen die Primärschlüssel - Fremdschlüsselbeziehung nicht in die Abfrage auf, können wir keine Abfragen auf mehrere Tabellen erstellen und dem zufolge die AuftragNr nicht anzeigen.
Die Lösung dieses Problems ist der Left-Join (bzw. Right-Join). Beim Left-Join werden alle Datensätze der Tabelle im “From-Teil“ angezeigt. Existiert ein oder existieren mehrere Datensätze mit gültiger Primärschlüssel - Fremdschlüsselbeziehung im “left join on“ werden diese Informationen zusätzlich angezeigt. Beim Right-Join ist es genau umgekehrt.

4.1.28 SQL-28: Left-Join

Bei dieser Abfrage werden auch die Kunden angezeigt welche keine Aufträge erteilt haben, somit also alle Kunden die in der Tabelle stehen. Wenn ein Kunde keinen Auftrag offen hat, entstehen dadurch bei der Ausgabe im Bereich der Auftragsdaten leeren Zellen in denen Null steht. Null bedeutet “kein Wert“ also auch kein leerer String oder ähnliches. Null bedeutet “nichts“. Ändert man nun die Abfrage und schreibt statt LEFT dann RIGHT, ändert sich das Ergebnis wie folgt:

SQL Statement

```
Select Kunde.Name, Auftrag.AuftragNr, Auftrag.Lieferdatum
from Auftrag, auftragProdukt, Produkt, Kunde
where Produkt.Name='USB-Stick (1GB)'
and Produkt.ProduktNr=auftragProdukt.ProduktNr
and auftragProdukt.AuftragNr=Auftrag.AuftragNr
and Auftrag.KundeNr=Kunde.KundeNr
```

Ergebnis

Meier AG	2	2015-10-09
Meier AG	9	2015-12-30
Meier AG	13	2015-12-29
Schulz GmbH	1	2015-10-10
Schulz GmbH	7	2015-12-20
Schulz GmbH	8	2015-12-27
Fischer	11	2015-12-24

Abbildung 4.28
SQL-26: Abfrage über vier Tabellen

SQL Statement

```
Select Kunde.Name, Kunde.PLZ, Kunde.Stadt, Auftrag.AuftragNr
from Kunde, Auftrag
where Kunde.KundeNr=Auftrag.KundeNr

oder in der alternativen Schreibweise

Select Kunde.Name, Kunde.PLZ, Kunde.Stadt, Auftrag.AuftragNr
from Kunde
join Auftrag on (Kunde.KundeNr=Auftrag.KundeNr)
```

Abbildung 4.29
SQL-27: Joins, weiteres Beispiel

4.1.29 SQL-29: Right-Join

Bei einem Right-Join werden nun alle Daten der Tabelle angezeigt, welche im “Right-Join“-Teil der Abfrage stehen. Sind dazu keine Verknüpfungen in der im “From“-Teil stehenden Tabelle vorhanden, (hier die Tabelle Auftrag) so werden die Zeilen mit NULL-Werten gefüllt.

4.1.30 SQL-30: IS NULL

Aufgabe:
Geben Sie die Kunden aus welche noch keine Aufträge erteilt haben.

Durch das Benutzen der Bedingung “is null“ erreichen wir, dass nur diejenigen Kunden angezeigt werden, welche noch keine Aufträge erteilt haben. Wir haben Ihnen beide Abfragen zur Verfügung gestellt, so dass Sie direkt die Unterschiede zwischen “Left-Join“ und “Right-Join“ erkennen können. Das Ergebnis ist bei beiden Abfragen identisch.

SQL Statement

```
SELECT * FROM Kunde
LEFT JOIN Auftrag
ON Kunde.KundeNr=Auftrag.KundeNr
```

Ergebnis

KundeNr	Name	PLZ	Stadt	KundeNr	AuftragNr	Auftragdatum	Lieferdatum
2	Meier AG	47543	Bochum	2	2	2015-10-08	2015-10-09
2	Meier AG	47543	Bochum	2	6	2015-12-09	2015-12-16
2	Meier AG	47543	Bochum	2	9	2015-12-09	2015-12-30
2	Meier AG	47543	Bochum	2	13	2015-12-09	2015-12-29
1	Schulz GmbH	44701	Bochum	1	1	2015-10-09	2015-10-10
1	Schulz GmbH	44701	Bochum	1	3	2014-11-12	2014-11-14
1	Schulz GmbH	44701	Bochum	1	5	2015-12-09	2015-12-17
1	Schulz GmbH	44701	Bochum	1	7	2015-12-09	2015-12-20
1	Schulz GmbH	44701	Bochum	1	8	2015-12-09	2015-12-27
1	Schulz GmbH	44701	Bochum	1	10	2015-12-09	2010-12-23
1	Schulz GmbH	44701	Bochum	1	12	2015-12-09	2015-12-22
3	Fischer	44787	Bochum	3	4	2015-10-09	2015-11-09
3	Fischer	44787	Bochum	3	11	2015-12-09	2015-12-24
4	Müller KG	45501	Essen	*NULL*	*NULL*	*NULL*	*NULL*
5	ZE GmbH	45501	Essen	*NULL*	*NULL*	*NULL*	*NULL*

Abbildung 4.30
SQL-28: Left-Join

4.2 Datensätze löschen

Wir haben jetzt die Statements zur Auswahl von Datensätzen durch. Der Rest ist jetzt relativ einfach. Zunächst das Löschen von Datensätzen.

- Das Löschen ist beschränkt auf eine Tabelle, Sie können also nicht wie beim Select mehrere Tabellen angeben.

- Es werden immer ganze Datensätze / Zeilen gelöscht (nicht einzelne Attribute / Spalten), deswegen werden hinter dem Befehl “Delete“ auch keine Spalten aufgeführt.

Achtung: Wird keine where-Klausel angegeben, werden alle Datensätze gelöscht. Ansonsten werden die ausgewählten Datensätze gelöscht (Semantik von where wie bei Select, eben nur mit einer Tabelle). Beispiele:

4.2.1 SQL-31: Alle Daten in einer Tabelle löschen

Alle Daten der Tabelle Kunde löschen:

SQL Statement

```
Select * from Auftrag
right join Kunde
on Kunde.KundeNr=Auftrag.KundeNr
```

Ergebnis

KundeNr	AuftragNr	Auftragdatum	Lieferdatum	KundeNr	Name	PLZ	Stadt
2	2	2015-10-08	2015-10-09	2	Meier AG	47543	Bochum
2	6	2015-12-09	2015-12-16	2	Meier AG	47543	Bochum
2	9	2015-12-09	2015-12-30	2	Meier AG	47543	Bochum
2	13	2015-12-09	2015-12-29	2	Meier AG	47543	Bochum
1	1	2015-10-09	2015-10-10	1	Schulz GmbH	44701	Bochum
1	3	2014-11-12	2014-11-14	1	Schulz GmbH	44701	Bochum
1	5	2015-12-09	2015-12-17	1	Schulz GmbH	44701	Bochum
1	7	2015-12-09	2015-12-20	1	Schulz GmbH	44701	Bochum
1	8	2015-12-09	2015-12-27	1	Schulz GmbH	44701	Bochum
1	10	2015-12-09	2010-12-23	1	Schulz GmbH	44701	Bochum
1	12	2015-12-09	2015-12-22	1	Schulz GmbH	44701	Bochum
3	4	2015-10-09	2015-11-09	3	Fischer	44787	Bochum
3	11	2015-12-09	2015-12-24	3	Fischer	44787	Bochum
NULL	*NULL*	*NULL*	*NULL*	4	Müller KG	45501	Essen
NULL	*NULL*	*NULL*	*NULL*	5	ZE GmbH	45501	Essen

Abbildung 4.31
SQL-29: Right-Join

4.2.2 SQL-32: Textbedingung beim löschen

In der Tabelle Kunde alle Kunden aus Essen löschen:

4.2.3 SQL-33: Mehrere Bedingungen beim löschen

In der Tabelle Kunde den Kunden löschen welcher die Kundennummer 1 hat und in Bochum wohnt.

Achtung: Was passiert denn mit den Bestellungen, wenn wir einen Kunden löschen? Bei manchen Datenbankmanagementsystemen können Sie bei der Definition der Beziehung zwischen den beiden Tabellen festlegen, dass hier ein Fremdschlüssel vorliegt. Das DBMS achtet dann darauf, dass zu einem Fremdschlüssel auch immer ein Datensatz mit dem entsprechenden Primärschlüssel vorhanden ist. Um festzulegen, was passiert, wenn Sie einen Datensatz löschen auf den mittels eines Fremdschlüssels verwiesen wird, haben Sie folgende Möglichkeiten:

- VETO
  Das Löschen eines Kunden wird verhindert, wenn es noch Bestellungen für ihn gibt

- SET NULL
  Der Fremdschlüssel in den Bestellungen des Kunden wird auf Null gesetzt, womit für diese Bestellungen keine Kunden mehr bekannt sind.

- DELETE CASCADE
  Die Bestellungen dieses Kunden werden mit gelöscht.

Damit sind wir auch durch das Löschen durch!

SQL Statement

```
Select * from Kunde
left join Auftrag
on Kunde.KundeNr=Auftrag.KundeNr
where Auftrag.KundeNr is null
```

Ergebnis

KundeNr	Name	PLZ	Stadt	KundeNr	AuftragNr	Auftragdatum	Lieferdatum
4	Müller KG	45501	Essen	NULL	NULL	NULL	NULL
5	ZE GmbH	45501	Essen	NULL	NULL	NULL	NULL

SQL Statement

```
Select * from Auftrag
right join Kunde
on Kunde.KundeNr=Auftrag.KundeNr
where Auftrag.KundeNr is null
```

Ergebnis

KundeNr	AuftragNr	Auftragdatum	Lieferdatum	KundeNr	Name	PLZ	Stadt
NULL	NULL	NULL	NULL	4	Müller KG	45501	Essen
NULL	NULL	NULL	NULL	5	ZE GmbH	45501	Essen

Abbildung 4.32
SQL-30: IS-NULL

SQL Statement

```
Delete from Kunde
```

Ergebnis

Gelöschte Zeilen: 1039 (die Abfrage dauerte 0.0007 sek.)

SQL-Befehl:

```
DELETE FROM Kunde
```

Abbildung 4.33
SQL-31: Alle Daten einer Tabelle löschen

4.3 Datensätze ändern

4.3.1 SQL-34: Alle Werte einer Spalte ändern

Alle Kunden sind plötzlich nach Duisburg umgezogen

4.3.2 SQL-35: Zahlenwert ändern

Die Postleitzahl stimmt noch nicht:

SQL Statement

```
Delete from Kunde where Ort="Essen"
```

Ergebnis

Gelöschte Zeilen: 2 (die Abfrage dauerte 0.0007 sek.)

SQL-Befehl:

```
DELETE FROM Kunde WHERE Ort = "Essen"
```

Vor Delete

KundeNr	Vorname	Nachname	PLZ	Ort
1	Otto	Mustermann	44801	Bochum
2	Hans	Dampf	44135	Dortmund
3	Hubert	Jäger	45359	Essen
4	August	Lustig	44805	Bochum
5	Gisela	Blumenfeld	44801	Bochum
6	Peter	Michael	44795	Bochum
7	Agnes	Blumenfeld	44789	Bochum
8	Willi	Fragenzapf	45219	Essen

Nach Delete

KundeNr	Vorname	Nachname	PLZ	Ort
1	Otto	Mustermann	44801	Bochum
2	Hans	Dampf	44135	Dortmund
4	August	Lustig	44805	Bochum
5	Gisela	Blumenfeld	44801	Bochum
6	Peter	Michael	44795	Bochum
7	Agnes	Blumenfeld	44789	Bochum

Abbildung 4.34
SQL-32: Löschen mit Einschränkung

4.3.3 SQL-36: Mehrere Werte eines Datensatzes ändern

Unser Mitarbeiter, Herr Mustermann, hat geheiratet und ist nach Hamburg gezogen:

4.3.4 SQL-37: Zahlenwerte berechnen

Preiserhöhung bei unseren Artikeln von 10 %:

Nicht schön, wir runden die Preise auf zwei Stellen nach dem Komma:

4.3.5 SQL-38: Runden von Zahlenwerten

SQL Statement

```
Delete from Kunde where Ort ="Bochum" and KundeNr=1
```

Ergebnis

Gelöschte Zeilen: 1 (die Abfrage dauerte 0.0007 sek.)

SQL-Befehl:

```
DELETE FROM Kunde WHERE Ort = "Bochum" AND KundeNr =1
```

Vor Delete

KundeNr	Vorname	Nachname	PLZ	Ort
1	Otto	Mustermann	44801	Bochum
2	Hans	Dampf	44135	Dortmund
3	Hubert	Jäger	45359	Essen
4	August	Lustig	44805	Bochum
5	Gisela	Blumenfeld	44801	Bochum
6	Peter	Michael	44795	Bochum
7	Agnes	Blumenfeld	44789	Bochum
8	Willi	Fragenzapf	45219	Essen

Nach Delete

KundeNr	Vorname	Nachname	PLZ	Ort
2	Hans	Dampf	44135	Dortmund
3	Hubert	Jäger	45359	Essen
4	August	Lustig	44805	Bochum
5	Gisela	Blumenfeld	44801	Bochum
6	Peter	Michael	44795	Bochum
7	Agnes	Blumenfeld	44789	Bochum
8	Willi	Fragenzapf	45219	Essen

Abbildung 4.35
SQL-33: Löschung mit Einschränkungen

SQL Statement

```
Update Kunde set Ort ="Duisburg"
```

Ergebnis

KundeNr	Vorname	Nachname	PLZ	Ort
2	Hans	Dampf	44135	Duisburg
3	Hubert	Jäger	45359	Duisburg
4	August	Lustig	44805	Duisburg
5	Gisela	Blumenfeld	44801	Duisburg
6	Peter	Michael	44795	Duisburg
7	Agnes	Blumenfeld	44789	Duisburg
8	Willi	Fragenzapf	45219	Duisburg

Abbildung 4.36
SQL-34: Alle Werte einer Spalte ändern

4.4 Daten einfügen

Hinzufügen neuer Datensätze erfolgt mit dem ”insert”-Statement: Sie können immer nur ganze Zeilen in eine Tabelle einfügen. Allerdings müssen nicht alle Spalten auch gefüllt werden. Sie können die Spalten, die Sie befüllen wollen, hinter dem Tabellennamen angeben. Wenn dies nicht erfolgt, werden alle Spalten in der Reihenfolge der Tabellendefinition verwendet. Spalten die hier nicht angegeben werden, bleiben leer. Bei den meisten Datenbankmanagementsysteme können Sie eine Spalte als “AutoIncrement“ bezeichnen. Diese Spalte wird dann durch die Datenbank selbst mit immer neuen, eindeutigen Werten befüllt. Auto-Increment-Einträge brauchen nicht angegeben zu werden (am besten lässt man die Spalte weg, alternativ kann auch ein DBMS-spezifischer Eintrag gemacht werden.)

SQL Statement

```
Update Kunde set PLZ ="47239"
```

Ergebnis

KundeNr	Vorname	Nachname	PLZ	Ort
2	Hans	Dampf	47239	Duisburg
3	Hubert	Jäger	47239	Duisburg
4	August	Lustig	47239	Duisburg
5	Gisela	Blumenfeld	47239	Duisburg
6	Peter	Michael	47239	Duisburg
7	Agnes	Blumenfeld	47239	Duisburg
8	Willi	Fragenzapf	47239	Duisburg

Abbildung 4.37
SQL-35: Zahlenwerte ändern

SQL Statement

```
Update Mitarbeiter set
            Nachname="Müller",
            Ort ="Hamburg",
            Telefon ="040-235488585"
where MitarbeiterNr=1
```

Ergebnis

MitarbeiterNr	Vorname	Nachname	Ort	Telefon
1	Otto	Müller	Hamburg	040-235488585
2	Hans	Dampf	Dortmund	0231-34561
3	Hubert	Jäger	Essen	0201-975835
4	August	Lustig	Bochum	0234-452395
5	Gisela	Blumenfeld	Bochum	0234-6593483
6	Peter	Michael	Bochum	0234-828492

Abbildung 4.38
SQL-36: Mehrere Spalten eines Datensatzes ändern

4.4.1 SQL-39: Einfügen von Datensätzen

SQL Statement

```
Update Artikel set Preis = 1.1 * Preis
```

Ergebnis

ArtikelNr	Name	Preis	Einheit
1	Laminat	9.35	qm
2	Tapete "Serengeti"	7.48	m
3	Klebeband	7.205	Rolle
4	Farbe Weiss	6.93	Liter
5	Tapetenkleister	4.62	Packung
6	Pinsel schmal	2.772	Stück
7	Rasendünger	6.743	kg

Abbildung 4.39
SQL-37: Ändern von Daten durch Neuberechnung

SQL Statement

```
Update Artikel set Preis = round(Preis, 2)
```

Ergebnis

ArtikelNr	Name	Preis	Einheit
1	Laminat	9.35	qm
2	Tapete "Serengeti"	7.48	m
3	Klebeband	7.2	Rolle
4	Farbe Weiss	6.93	Liter
5	Tapetenkleister	4.62	Packung
6	Pinsel schmal	2.77	Stück
7	Rasendünger	6.74	kg

Abbildung 4.40
SQL-38: Runden von Zahlenwerten

4.4.2 SQL-40: Mehrere Zeilen einfügen

4.4.3 SQL-41: Auto increment

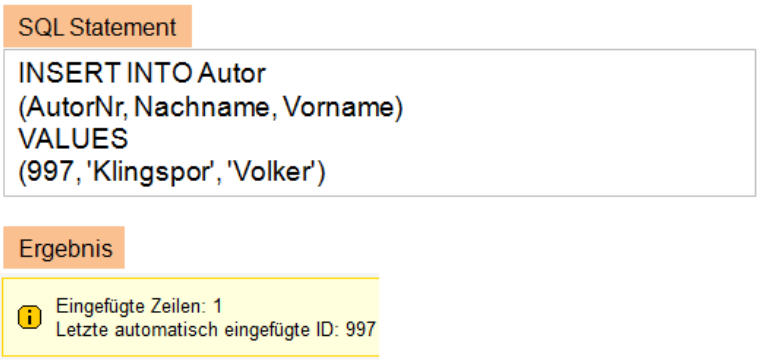


Abbildung 4.41
SQL-39: Einfügen eines Datensatzes

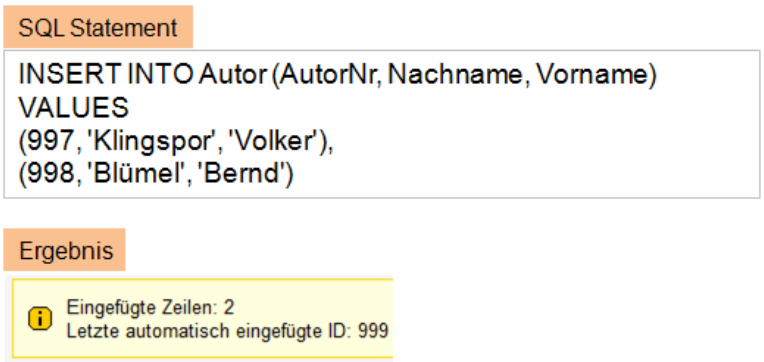


Abbildung 4.42
SQL-40: Mehrere Datensätze einfügen

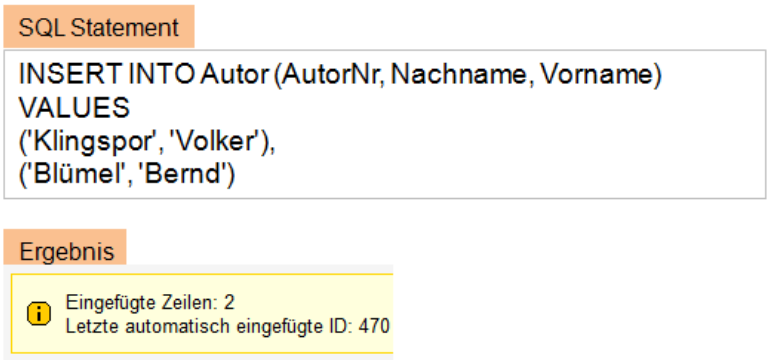


Abbildung 4.43
SQL-41: auto-increment Funktion

Kapitel 5

phpmyadmin

phpMyAdmin ist eine Anwendung zur Verwaltung von MySQL-Datenbanken welche im Browser ausgeführt wird. phpMyAdmin kann somit nur über den Webbrowser benutzt werden. Zum “Starten“ von phpMyAdmin muss ein bestimmter URL, unter dem die Anwendung läuft, in den Browser eingegeben werden. Damit phpMyAdmin auf dem Server ausgeführt werden kann, sind folgende drei Anwendungen nötig:

1. ein Webserver,

2. eine PHP-Installation auf dem Webserver und

3. eine MySQL-Datenbank.

Fehlt eine dieser drei Anwendungen, läuft phpMyAdmin nicht. Unter http://www.easyphp.org/ finden Sie ein Installationspaket, mit welchem Sie per Mausklick die erforderlichen Anwendungen zur Nutzung von phpMyAdmin auf Ihrem Windows-Rechner installieren können. Unter http://www.phpmyadmin.net finden Sie mehr Informationen zu phpMyAdmin.

5.1 Weboberfläche aufrufen

Zunächst rufen Sie phpMyAdmin auf dem Hochschulserver unter folgendem URL auf:
Den Usernamen sowie das Passwort zum Login erhalten Sie in der Veranstaltung.

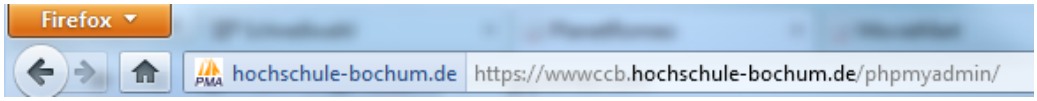


Abbildung 5.1
Aufruf der Weboberfläche

5.2 Datenbank anlegen

Eine neue Datenbank ”weihnachten” wird angelegt:

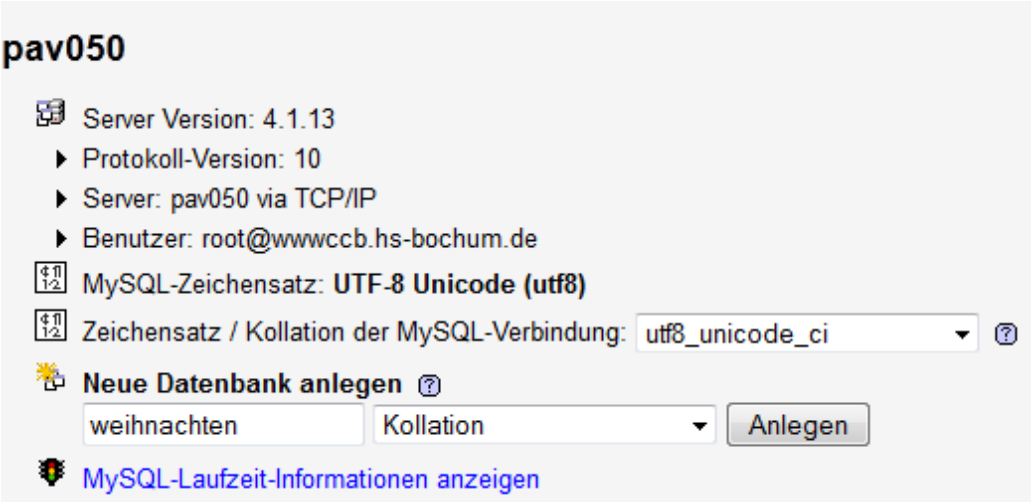


Abbildung 5.2
Neue Datenbank anlegen

Abbildung 5.3
Datenbank erfolgreich angelegt

5.3 Tabelle anlegen

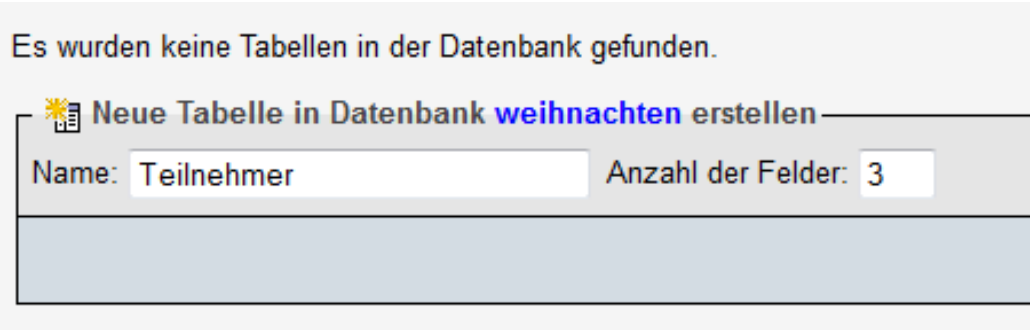


Abbildung 5.4
Neue Tabelle mit drei Spalten anlegen

5.4 Spalten sowie Datentypen definieren

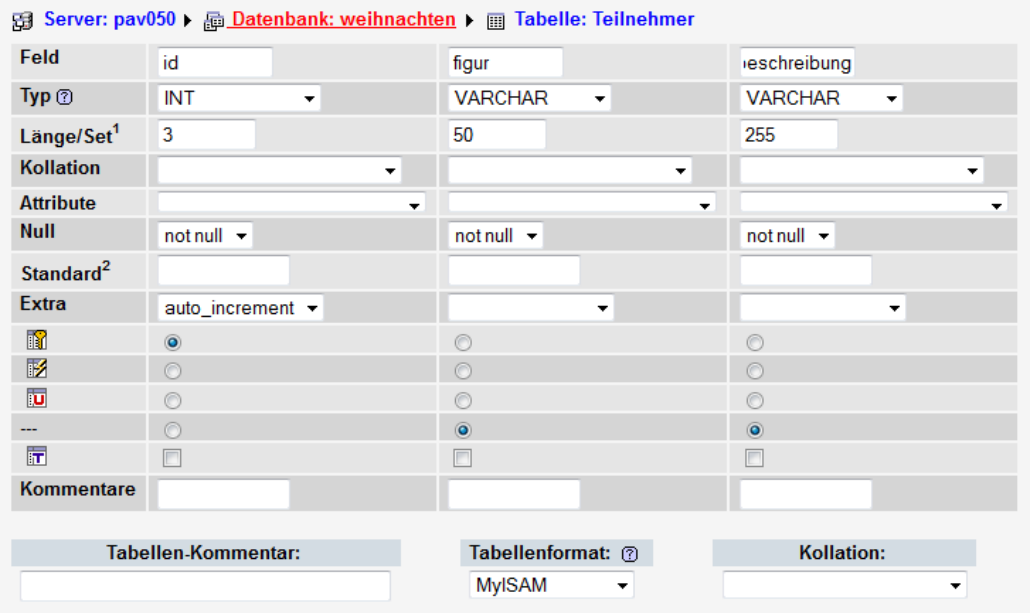


Abbildung 5.5
Spalten und Attribute definieren

5.5 Daten in die Tabelle einfügen

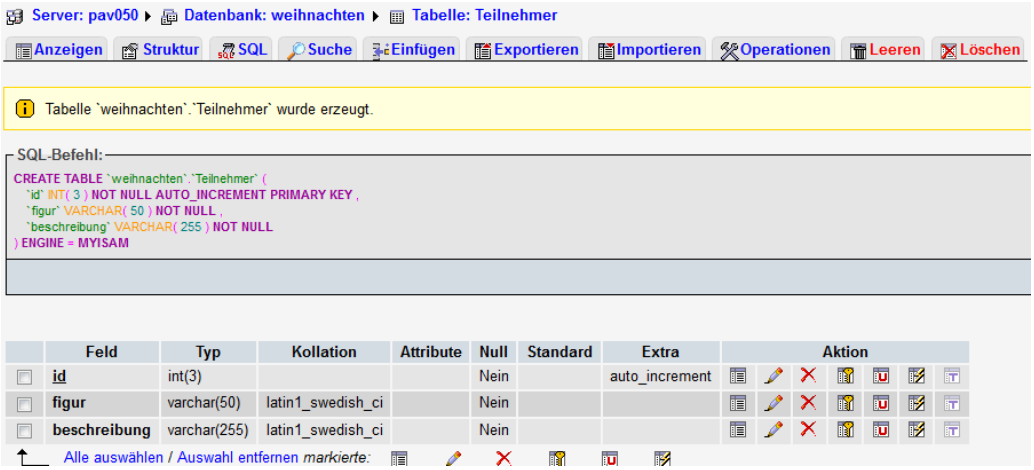


Abbildung 5.6
Tabelle erfolgreich angelegt

Abbildung 5.7
Daten in die Tabelle einfügen

5.6 Daten der Tabelle anzeigen

Abbildung 5.8
Daten erfolgreich eingetragen

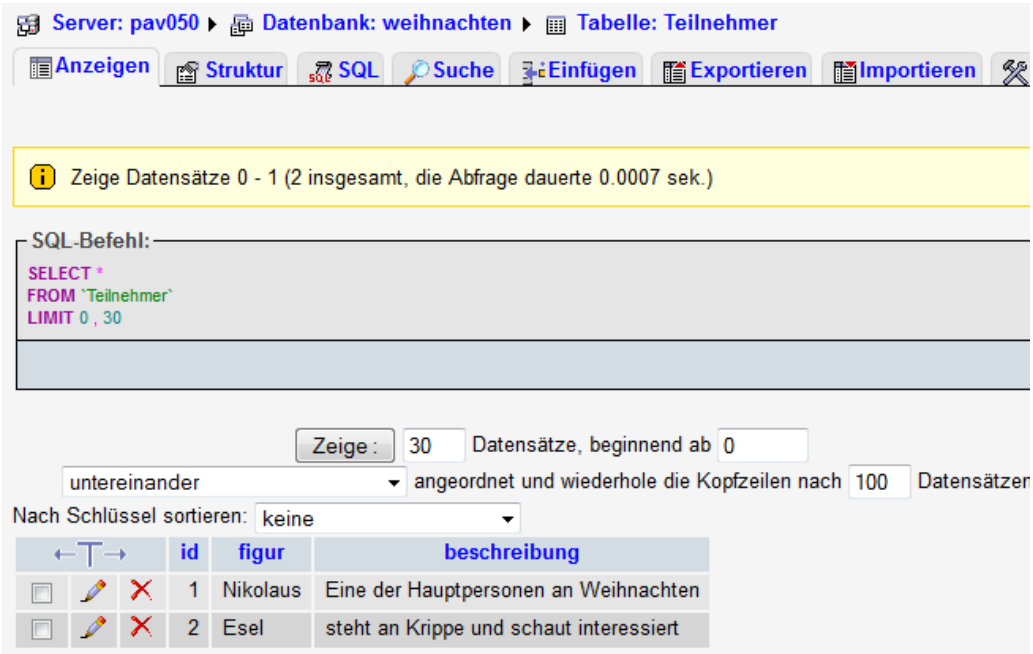


Abbildung 5.9
Alle Daten in der Tabelle anzeigen

5.7 SQL-Abfrage definieren

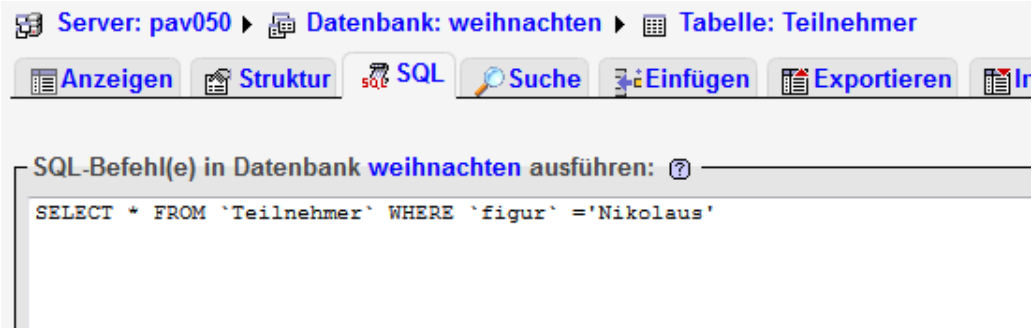


Abbildung 5.10
SQL-Abfrage definieren

Abbildung 5.11
Abfrageergebnis anzeigen

Stichwortverzeichnis