Datenbanken

ERM, Tabellen aus dem ERM, SQL

Bernd Blümel

Version: 21. Februar 2005

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Wie man es nicht machen sollte

Eine Datenbank zu erstellen, ist keine triviale Aufgabe. Ich möchte zunächst anhand eines Beispiels Problematiken bei dem Design einer Datenbank aufzeigen.

KundeNr	Name	PLZ	Stadt	AuftragNr	AuftragDatum	AuftragBeschreibung
1	Schulz GmbH	44701	Bochum	1	21.01.2005	Werkhalle streichen
2	Meier AG	47543	Bochum	2	22.01.2005	Teppichboden verlegen
1	Schulz GmbH	44701	Bochum	3	24.01.2005	Laminat verlegen
3	Fischer	44787	Bochum	4	27.01.2005	Tapezieren
1	Schulz GmbH	44702	Bochum	5	01.02.2005	Werkhalle Boden verlegen
2	Meier GmbH	47543	Bochum	6	01.02.2005	Laminat verlegen

Abbildung 1.1: Nicht wirklich gutes Design einer Datenbank

Die in Abb. 1.1 dargestellte Datenbank ist für einen typischen Betriebswirt :-)) eigentlich ziemlich natürlich. So würde man es machen, wenn man Aufträge handschriftlich erfaßt oder eine Tabellenkalkulation wie Excel oder OpenOffice Calc benutzt. Wir können jedoch sofort einige Probleme identifizieren:

- Die Datenbank enthält redundante (überflüssige) Daten. Die Daten der Kunden mit mehreren offenen Aufträgen werden auch mehrfach abgespeichert. Dies verschenkt Speicherplatz.
- Die Datenbank enthält widersprüchliche Daten. Im dritten und fünften Datensatz hat der Kunde Schulz unterschiedliche Postleitzahlen. Hier liegt offensichtlich ein Eingabefehler vor. Im zweiten und sechsten Datensatz hat der Kunde Meier unterschiedliche Rechtsformen. Dies kann ein Tippfehler sein, der Kunde Meier kann aber seine Rechtsform auch geändert haben.

Es gibt aber auch noch weitere nicht sofort ersichtliche Problematiken:

- Die Daten eines Kunden verschwinden, wenn der letzte Auftrag des Kunden gelöscht wird (Löschanomalie).
- Einen neuen Kunden kann man nur dann eintragen, wenn er einen Auftrag erteilt (Einfügeanomalie).
- Wenn man Daten eines Kunden ändern will, so muss man das so häufig machen, wie dieser Kunde Aufträge in der Datenbank hat (Änderungsanomalie).

Alle Probleme resultieren aus einem Tatbestand. Die Begriffe Datenbank und Tabelle werden nicht getrennt. In Wirklichkeit wollen wir ja Informationen zu Kunden und zu Aufträgen speichern. Und das sind sehr unterschiedliche Objekte des wirklichen Lebens. Wir speichern alle Informationen zu zwei unterschiedlichen Objekten in einer Tabelle und das ist der Fehler.

Anders als in einer Tabellenkalkulation kann man in einer Datenbank mehrere Tabellen anlegen und diese Tabellen mit einander verbinden. Ich zeige jetzt eine richtige Lösung unserer Problematik:

Wir erzeugen eine Tabelle Kunde und eine Tabelle Auftrag (vgl. Abb. 1.2).

In der Tabelle Kunde verwalten wir die Daten der Kunden. Da wir hier für jeden Kunden genau einen Eintrag haben, fallen alle oben beschriebenen Problematiken weg. Da Kunde und Auftrag entkoppelt sind, können Kunden auch ohne Aufträge aufgenommen werden. Das Löschen von Aufträgen hat keinen Einfluß auf die Kundentabelle.

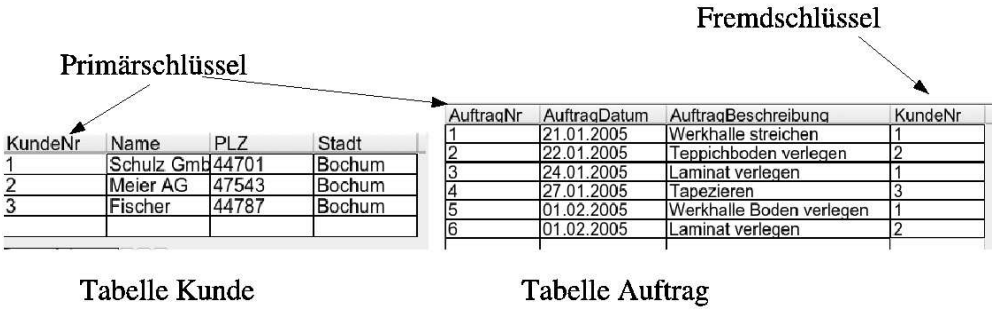


Abbildung 1.2: Datenbank mit optimierter Struktur

In der Kundentabelle haben wir ein besonderes Feld: Das Feld KundeNr. Dies ist, wie auch in Abb. 1.2 dargestellt, der Primärschlüssel der Tabelle. Ein Primärschlüssel identifiziert eindeutig einen Datensatz einer Tabelle. Oder anders herum ausgedrückt: Es gibt keine zwei Datensätze mit demselben Primärschlüssel.

In der Auftragstabelle speichern wir die Informationen zu den Aufträgen. Auch diese Tabelle verfügt über einen Primärschlüssel, es ist das Feld AuftragNr. Zum Abschluss müssen wir eine Beziehung zwischen den Tabellen Kunde und Auftrag herstellen. Hierzu dient das Feld KundeNr in der Tabelle Auftrag. Wie Sie sich sicher erinnern (steht ja nun im Absatz drüber) ist KundeNr der Primärschlüssel der Tabelle Kunde, identifiziert also eindeutig einen Datensatz in der Tabelle Kunde. KundeNr ist nicht Primärschlüssel der Tabelle Auftrag, denn es gibt ja in dieser Tabelle durchaus mehrere Datensätze mit der gleichen Kundennummer. Wir benötigen die KundeNr in Auftrag, um Adresse, Namen usw. des Kunden zu ermitteln, der den Auftrag erteilt hat. Ein Datenbanksystem kann mit der in der Tabelle Auftrag eingetragenen KundenNr den Datensatz des Kunden in der Tabelle Kunde ermitteln. Das Feld KundeNr in der Tabelle Auftrag hat ebenfalls eine besondere Bezeichung: Fremdschlüssel. Ein Fremdschlüssel allgemein ist ein Feld einer Tabelle, welches Primärschlüssel einer anderen Tabelle ist. Löschen eines Datensatzes in Auftrag löscht also nur die Beziehung zum zugehörigen Datensatz in Kunde, hat aber keine Auswirkung auf die Tabelle Kunde.

Übrigens ist auch die Richtung der Verknüpfung zweier Tabellen wichtig. Hätten wir AuftragNr, den Primärschlüssel von Auftrag, als Fremdschlüssel in die Tabelle Kunde übernommen, hätten wir sofort vor einem neuen Problem gestanden: Was machen wir mit Kunden, die mehrere Aufträge offen haben? Fügen wir mehrere Felder mit AuftragNr an, also Auftrag1Nr, Auftrag2Nr und so weiter? Das ist schlecht, weil diese Felder für Kunden, die gerade keine Aufträge offen haben, nicht gefüllt sind. Andererseits müssen wir die Tabellenstruktur ändern, wenn der erste Kunde mehr Aufträge erteilt, als gerade in der Tabelle Kunde vorgesehen.

Bei der in Abb. 1.2 dargestellten Struktur existiert diese Problematik nicht, denn jedem Auftrag ist genau ein Kunde zugeordnet, dessen KundeNr wir in Auftrag übernehmen.

Kapitel 2

Tabellen, Felder, Datensätze und Schlüssel

Wie bereits in Kapitel 1 dargestellt, werden die Daten, die in einer Datenbank gespeichert werden sollen, auf Tabellen aufgeteilt. Abb. 2.1 zeigt beispielhaft einen Teil der Tabellen der unserem Intranet-Informationssystem zugrunde liegenden Datenbankstruktur.

Tabelle	Aktion	Einträge	Typ
assignmentBachelorsemesterDiplomsemestergroup		6	MyISAM
bachelorDiplom		39	MyISAM
blockEvent		113	MyISAM
cancellation		0	MyISAM
cancellationTemp		0	MyISAM
curriculum		94	MyISAM
day		7	MyISAM
employee		121	MyISAM
employeeLecture		130	ISAM
employeePasswordDistributor		300	MyISAM
exam		1.288	MyISAM
importTimetable		387	MyISAM
lecture		225	MyISAM

Abbildung 2.1: Ausschnitt der Tabellenstruktur des Intranet-Informationssystems

Tabellen bestehen aus Spalten und Zeilen[1]. Im Datenbank-Kontext heißen die Spalten Attribute oder Felder. Sie legen fest, welche Informationen in der Tabelle abgespeichert werden. In Abb. 2.2 existieren die Felder (oder Attribute)[2] KundeNr, Name, PLZ und Stadt. Dies sind die Informationen, die wir zu unseren Kunden abspeichern wollen. Die Zeile einer Tabelle heißen Datensätze. Ein Datensatz enthält also konkrete Werte für die Felder eines Kunden.

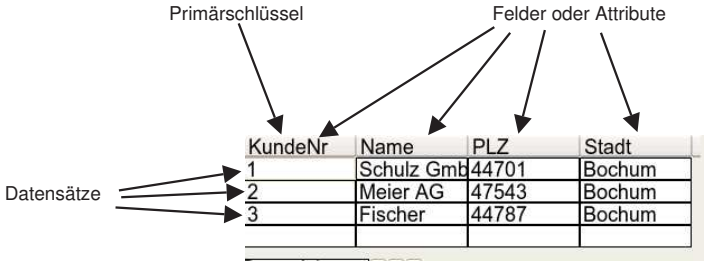


Abbildung 2.2: Die Tabelle Kunde

Jede Tabelle besitzt einen Primärschlüssel[3]. Wie ebenfalls bereits in Kapitel 1 dargestellt, identifiziert ein Primärschlüssel

[1] Wie Sie ja wahrscheinlich aus Excel wissen.

[2] Feld und Attribut werde ich von nun an synonym benutzen.

[3] Einige Datenbanksysteme, wie z.B. auch Access, erlauben es, Tabellen ohne Primärschlüssel anzulegen. Das machen wir nie, weil Datenbanken dadurch sehr langsam werden.

eindeutig einen Datensatz einer Tabelle. Oder anders herum ausgedrückt: Es gibt keine zwei Datensätze mit demselben Primärschlüssel. Der Primärschlüssel kann aus einem Feld bestehen (die KundeNr der Tabelle Kunde), es gibt aber auch Tabellen, mit zusammengesetzten Primärschlüsseln, was bedeutet, das der Primärschlüssel aus mehreren Feldern besteht. Details und Beispiele dazu werden Sie aber erst in Kapitel 4.2.2 lernen.

Wenn unter den Feldern der Tabelle kein Primärschlüsselkandidat ist (wie z.B. die ISBN-Nummer einer Tabelle Buch), erzeugen wir einen künstlichen Primärschlüssel, der einfach aus einer Zahl besteht, die wir für jeden neuen Datensatz um Eins hochzählen (so ist es ja auch bei der Tabelle Kunde in Abb. 2.2).

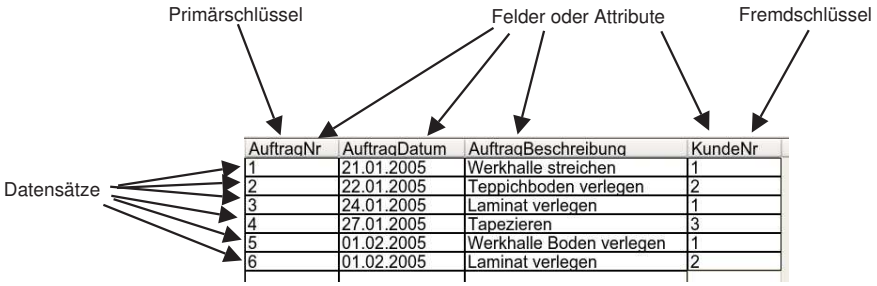


Abbildung 2.3: Die Tabelle Auftrag

Abb. 2.3 zeigt die oben beschriebenen Begriffsbildungen für die Tabelle Auftrag. Hier kommt eine weitere Begrifflichkeit hinzu: Der Fremdschlüssel.

Das Feld KundeNr, das ja nun Primärschlüssel der Tabelle Kunde ist (vgl. Abb. 2.2), ist ebenfalls ein Feld der Tabelle Auftrag. Hier ist KundeNr kein Primärschlüssel, die Aufträg 1, 3 und 5 sind alle Aufträge der Schulz GmbH (KundeNr 1), identifiziert also nicht eindeutig einen Datensatz der Tabelle Auftrag. Primärschlüssel der Tabelle Auftrag ist AuftragNr. Das Feld KundeNr der Tabelle Auftrag ist ein Fremdschlüssel. Ein Fremdschlüssel allgemein ist ein Feld einer Tabelle (hier der Tabelle Aufrag), welches Primärschlüssel einer anderen Tabelle (hier der Tabelle Kunde) ist. Fremdschlüssel benötigen wir, um die in der Datenbank in unterschiedlichen Tabellen abgespeicherten Informationen, wieder zusammenzufügen. Wir können über die Primärschlüssel-Fremdschlüssel-Kombination Fragen beantworten wie: Welche Kunden (Name soll hier ausgegeben werden) haben im Februar 2005 Aufträge erteilt? Das Datenbanksystem findet die Aufträge (es sind die Aufträge 5 und 6) in der Tabelle Auftrag. Mit der zum Datensatz gehörenden KundeNr (bei Auftrag 5 ist es die 1, bei Auftrag 6 die 2) kann das Datenbanksystem nun in der Tabelle Kunde den zugehörigen Datensatz ermitteln (da kann es jeweils nur einen geben, denn KundeNr ist in Kunde ja Primärschlüssel). Das Datenbanksystem ermittelt so die zugehörigen Kundennamen Schulz GmbH und Meier AG.

Kapitel 3

Entity Relationship Modellierung

In Kapitel 1 habe ich eine Tabellenstruktur für ein einfaches Beispiel vorgestellt. Es stellt sich die Frage, wie wir eine optimale Tabellenstruktur für eine gegebene Problemstellung entwickeln können. Die Lösung dazu ist die Entity-Relationship Modellierung. Hier erstellen wir zunächst ein Modell aus der gegebenen Problemstellung. Aus dem Modell können wir dann mit Standardmethodiken (dies wird in Kapitel 4 beschrieben) eine optimale Tabellenstruktur herleiten.

In diesem Kapitel lernen Sie, wie man aus einer gegebenen Problemstellung zum Entity-Relationship Modell[1] gelangt. Zur Veranschaulichung benutze ich drei Beispiele:

Beispiel 3.1 *Erweiterung des Beispiels aus Kapitel 1*

Kunden können Aufträge erteilen, Aufträge enthalten Artikel.

Beispiel 3.2 *Ausschnitt aus dem Intranet-Informationssystem des Fachbereichs*

In den Veranstaltungsräumen der FH sind Ressourcen installiert. Damit sind Beamer, Tafeln, PC's usw. gemeint. Räume besitzen darüber hinaus eine Kapazitätsklasse. Damit ist eine Klassifizierung in Bezug auf ihre Anzahl an Sitzplätzen gemeint. Darüberhinaus führen Professoren der FH Veranstaltungen in festgelegten Zeitblöcken an Wochentagen in den Vorlesungsräumen durch. Die Benutzer des Informationssystems sind die Professoren und Studenten.

Beispiel 3.3 *Abbildung der Organisation eines Unternehmens (Ausschnitt)*

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

3.1 Entities (Das E in ERM)

Der erste Schritt bei der Entity-Relationship Modellierung ist die Ermittlung der Entities. Dies sind die Objekte der Realität, die durch die Datenbank abgebildet werden sollen. In Beispiel 3.1 sind dies

- Kunde
- Auftrag
- Artikel

In Beispiel 3.2 hingegen

- Raum
- Ressource
- Kapazitätsklasse

[1] Von nun an ERM abgekürzt.

- Veranstaltung
- Zeit
- Tag
- Benutzer
- Professor
- Student
- Semestergruppe

In Beispiel 3.3 sind dies

- Mitarbeiter
- Abteilung

ERM ist eine grafische Methode, daher werden die Entities grafisch dargestellt. Das Symbol für Entities ist das Rechteck. Abb. 3.1 zeigt die grafische Darstellung der Entities für Beispiel 3.1, Abb. 3.2 für Beispiel 3.2 und Abb. 3.3 für Beispiel 3.3.

Abbildung 3.1: Erstes ERM von Beispiel 3.1

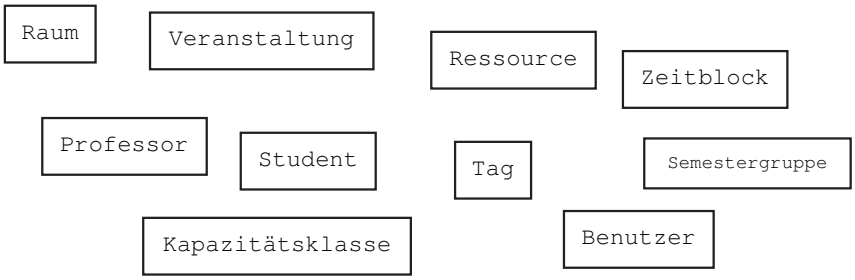


Abbildung 3.2: Erstes ERM von Beispiel 3.2

Abbildung 3.3: Erstes ERM von Beispiel 3.3

Wir unterscheiden Entities und schwache Entities oder Attribute. Attribute gehören zu den Entities und müssen selber nicht modelliert werden. Sie können in unserem Modell nicht eigenständig existieren. Ein Beispiel hierzu ist der Name eines Kunden. Der Name eines Kunden beschreibt einen Kunden in unserem Modell und kommt nur innerhalb des Kontextes des Kunden (in Beispiel 3.1) vor. Attribute modellieren wir nicht innerhalb unseres ERM, wir ermitteln sie, wenn

wir, wie Sie es in Kapitel 4 lernen werden, die Tabellen aus dem ERM ableiten. Hier überlegen wir uns nur: Handelt es sich um eine Entity oder um ein Attribut, das zu einer Entity gehört[2].

Ob es sich bei den durch die Datenbank zu beschreibenden Objekten um Entities oder Attribute handelt, ist allerdings nicht immer so leicht zu entscheiden, wie in obigem Fall. Ein gutes Beispiel dazu ist die Kapazität eines Raumes in Beispiel 3.2. Möchten wir nur die Anzahl Sitzplätze eines Vorlesungsraums in der Datenbank verwalten, so handelt es sich dabei sicherlich nicht um eine Entity, sondern einfach um ein Attribut der Entity Raum. Möchten wir hingegen den Vorlesungsräumen Kapazitätsklassen zuordnen, so z.B.:

- 30 Plätze, geeignet für normale Seminare des Hauptstudiums
- 30-50 Plätze, geeignet für größere Seminare des Hauptstudiums
- 50-70 Plätze, geeignet für eine Semestergruppe des Grundstudiums
- 70-100 Plätze, geeignet für zwei Semestergruppen des Grundstudiums,

dann macht es durchaus Sinn eine Entity Kapazitätsklasse einzuführen.

3.2 Beziehungen zwischen den Entities (Relations Das R in ERM)

Der nächste Schritt ist, die Beziehungen zwischen den Entities zu entdecken und zu modellieren. Einfache Beziehungen sind z.B.:

- Eine Beziehung zwischen Kunde und Auftrag in Beispiel 3.1 ist: Kunde erteilt Auftrag.
- Eine Beziehung zwischen Raum und Ressource in Beispiel 3.2 ist: Raum besitzt Ressource.

Nachdem oder besser während wir die Beziehungen zwischen den Entities ermitteln, müssen wir die Beziehungen klassifizieren.

3.2.1 1:n-Beziehung

Betrachten wir die Beziehung Kunde erteilt Auftrag. Hier können wir feststellen: Ein Kunde kann mehrere Aufträge erteilen, ein Auftrag kann aber nur von einem Kunden erteilt werden. Wir haben hier ein Beispiel einer 1:n-Beziehung. Dies wird häufig auch als Kardinalität der Beziehung bezeichnet. Beziehungen werden, wie in Abb. 3.4 gezeigt, in unserem Modell dargestellt.

Abbildung 3.4: ERM: Kunde-Auftrag

Beziehungen werden also durch Rauten dargestellt, die mit den Entities, zwischen denen die Beziehung besteht, verbunden werden. Jede Beziehung erhält einen Namen, unsere heißt „erteilt“. Der Name der Beziehung erscheint innerhalb der Raute. Die Kardinalitäten der Beziehung werden an die Verbindungslinien zu den Entities geschrieben.

Schauen wir uns nun die Beziehung zwischen Raum und Kapazitätsklasse an. Hier gilt: Ein Raum besitzt eine Kapazitätsklasse, eine Kapazitätsklasse kann aber mehreren Räumen zugeordnet sein. Wir haben also auch hier eine 1:n-Beziehung. Sie ist in Abb. 3.5 dargestellt.

Die allgemeine Definition der 1:n-Beziehung ist:
Einem Wert der ersten Entity (ein Kunde) sind mehrere Werte der zweiten Entity (mehrere Aufträge) zugeordnet, einem Wert der zweiten Entity (ein Auftrag) jedoch nur ein Wert der ersten Entity (ein Kunde).

[2]In anderen Lehrbüchern wird das durchaus anders gesehen und vorgeschlagen, die Attribute bereits während der Ermittlung der Entities mit zu modellieren. Ich finde dies allerdings ein wenig unübersichtlich.

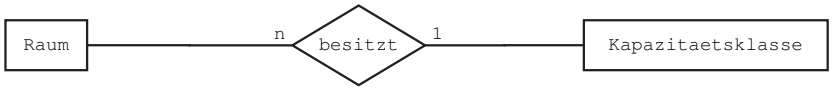


Abbildung 3.5: ERM: Raum-Kapazitätsklasse

3.2.2 n:m-Beziehung

Auch den Begriff der n:m-Beziehung leiten wir anhand von Beispielen her. Betrachten wir eine weitere Beziehung aus Beispiel 3.1, nämlich die Beziehung zwischen Aufträgen und Produkten. Hier gilt: Ein Auftrag kann mehrere Produkte enthalten, ein Produkt kann in mehreren Aufträgen vorkommen. Eine solche Beziehung heißt n:m-Beziehung und wird, wie in Abb. 3.6 gezeigt, in unserem Modell dargestellt.

Abbildung 3.6: ERM: Auftrag-Produkt

Beachten Sie: Die Entität, von der die Betrachtung ausgeht, wird immer im Singular verwendet:

- **Ein Auftrag** kann mehrere Produkte enthalten, **ein Produkt** 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.

Betrachen wir auch hier ein zweites Beispiel, die Beziehung zwischen Raum und Ressource aus dem Intranet. 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). Die grafische Darstellung zeigt Abb. 3.7.

Abbildung 3.7: ERM: Raum-Ressource

Die allgemeine Definition der n:m-Beziehung ist:
Einem Wert der ersten Entity (ein Auftrag) sind mehrere Werte der zweiten Entity (mehrere Produkte) zugeordnet, einem Wert der zweiten Entity (ein Produkt) ebenfalls mehrere Werte der ersten Entity (mehrere Aufträge).

3.2.3 1:1-Beziehung

Den Begriff 1:1-Beziehung sollten Sie nach dem bereits vorgestellten Stoff eigentlich selbst mit Leben füllen können. Dennoch erkläre ich auch diesen Beziehungstyp anhand eines Beispiels. Betrachten wir die Beziehung zwischen Abteilung und Mitarbeiter aus Beispiel 3.3. Hier gilt: Ein Mitarbeiter kann eine Abteilung leiten, eine Abteilung wird von einem Mitarbeiter geleitet. Dies ist eine 1:1-Beziehung und wird, wie in Abb. 3.8 gezeigt, in unserem Modell dargestellt.

Dieses Beispiel zeigt aber auch, dass es durchaus mehrere Beziehungen zwischen zwei Entities geben kann. Denn „Mitarbeiter arbeitet in Abteilung“ ist sicherlich auch eine Beziehung in unserem Modell. Es ist aber eine völlig andere Beziehung als „Mitarbeiter leitet Abteilung“. Daher muss auch diese Beziehung modelliert werden. Hier gilt: Ein Mitarbeiter arbeitet in einer Abteilung, in einer Abteilung können mehrere Mitarbeiter arbeiten. Es handelt sich also um eine 1:n-Beziehung. Sie ist in Abb. 3.9 dargestellt.

Abbildung 3.8: ERM: Mitarbeiter leitet Abteilung

Abbildung 3.9: ERM: Mitarbeiter arbeitet in Abteilung

Die allgemeine Definition der 1:1-Beziehung ist:
Einem Wert der ersten Entity (ein Mitarbeiter) ist ein Wert der zweiten Entity (eine Abteilung) zugeordnet, einem Wert der zweiten Entity (eine Abteilung) ebenfalls ein Wert der ersten Entity (ein Auftrag).

3.2.4 Beziehungen zwischen mehr als 2 Entities, Grad der Beziehung

Beziehungen zwischen mehr als 2 Entities sind ebenfalls möglich. Betrachten wir hierzu den Stundenplan aus unserem Intranet-Beispiel. Hier gilt: Ein Dozent kann mehrere Veranstaltungen in mehreren Räumen zu unterschiedlichen Zeiten an unterschiedlichen Wochentagen für unterschiedliche Semestergruppen durchführen. Eine Veranstaltung kann an unterschiedlichen Wochentagen von unterschiedlichen Dozenten in mehreren Räumen zu unterschiedlichen Zeiten für unterschiedliche Semestergruppen durchgeführt werden.

Hier haben wir sechs Entities, die an einer Beziehung beteiligt sind. Abb. 3.10 zeigt die Darstellung einer solchen Beziehung in unserem Modell.

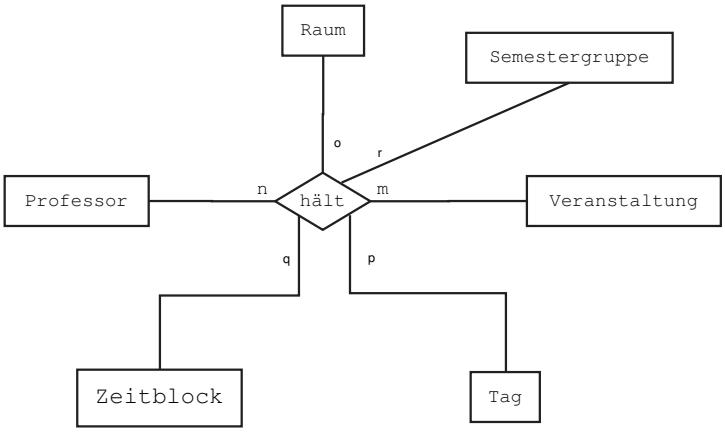


Abbildung 3.10: ERM: Stundenplan

3.2.5 Rekursive Beziehung, Beziehung einer Entity mit sich selbst

Entities können mit sich selbst in Verbindung stehen. Hierzu betrachen wir die Beziehung „Mitarbeiter ist verheiratet mit Mitarbeiter“. Hier steht die Entity Mitarbeiter mit sich selbst in Beziehung. Darüber hinaus ist es eine 1:1-Beziehung[3]. Abb. 3.11 zeigt die zwei Möglichkeiten, rekursive Beziehungen im ERM darzustellen.

[3]Weil Polygamie bei uns ja verboten ist.

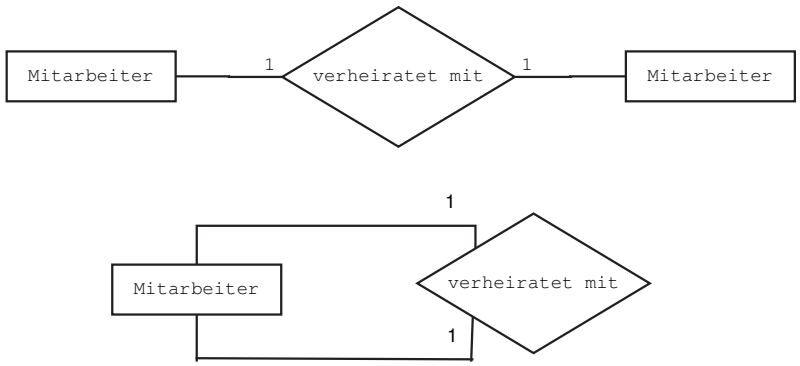


Abbildung 3.11: ERM: Rekursive Beziehung

3.2.6 Kann-Muss-Beziehung

Hier stellen wir fest, ob die Teilnahme einer Entity an einer Beziehung obligatorisch ist, oder nicht. Auch dies machen wir uns an unserer Kunden-Auftrag-Beziehung klar. Wir stellen fest: Ein Auftrag muss von einem Kunden kommen, oder anders ausgedrückt, Aufträge ohne Kunden kann es nicht geben. Andererseits müssen Kunden keine Aufträge erteilen, es kann sogar durchaus Kunden in unserer Datenbank geben, die noch nie einen Auftrag erteilt haben, die z.B Interesse an Produkten gezeigt haben und die wir deswegen mit Werbemaßnahmen beglücken wollen.

Die Fragestellungen, die wir hier klären, lauten also:

- Ein Auftrag, muss der von einem Kunden kommen? Wir beantworten dies mit ja.
- Ein Kunde, muss der einen Auftrag erteilen? Wir beantworten dies mit nein.

Muss wird im Modell durch einen Strich symbolisiert, kann durch einen Kreis. Wenn wir nicht wissen, ob es sich um eine Kann- oder Muss-Beziehung handelt, lassen wir das Symbol weg. Abb. 3.12 zeigt die grafische Darstellung im ERM.

Abbildung 3.12: ERM: Kunde-Auftrag mit Kann-Muss

Veranschaulichen wir uns den gleichen Sachverhalt anhand eines zweiten Beispiels, der Beziehung zwischen Auftrag und Produkt. Wir fragen zunächst: In einem Auftrag muß da ein Produkt vorkommen und beantworten die Frage mit Ja. Es kann keinen Auftrag ohne Produkte geben. Danach fragen wir uns: Ein Produkt, muss das in einem Auftrag vorkommen und beantworten die Frage mit Nein. Es kann ja durchaus Produkte geben, die wir bereits in der Datenbank erfasst haben, aber die noch so neu sind, dass sie noch nicht bestellt wurden. Abb. 3.13 zeigt die grafische Darstellung im ERM.

Abbildung 3.13: ERM: Produkt-Auftrag mit Kann-Muss

3.2.7 Generalisierung - Spezialisierung, Is-a-Beziehung

Die letzte Art der Beziehung, die wir abbilden müssen, ist die Generalisierung - Spezialisierung. Dies machen wir uns wieder am Beispiel unseres Intranets klar: Unser Intranet-Informationssystem verfügt über zwei Arten von Benutzern,

die Dozenten und Studenten. Diese Benutzer haben viel gemeinsam, nämlich Namen, Vornamen, email-Adresse, Handy-Nummer und so weiter. Sie unterscheiden sich aber auch, Studenten haben eine Matrikelnummer, einen Studiengang, Anzahl Semester, dies alles sind keine Eigenschaften von Professoren. Professoren wiederum verfügen über einen Raum an der FH, haben eine Sprechstunde, eine interne Telefonnummer, ein Lehrgebiet, alles keine Eigenschaften von Studenten. Wir sagen, es gibt eine Generalisierung, die Benutzer, mit zwei Spezialisierungen, den Professoren und Studenten. Abb. 3.14 zeigt die Darstellung einer solchen Beziehung in unserem Modell.

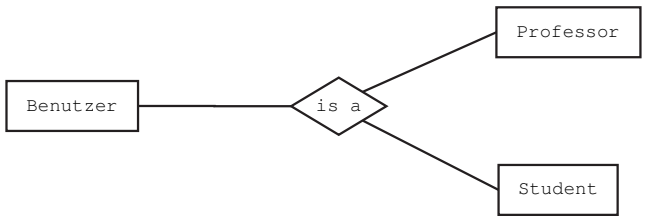


Abbildung 3.14: ERM: Stundenplan

3.2.8 Redundante Beziehungen

Um Redundanzen in den Daten zu vermeiden, dürfen wir keine Beziehungen modellieren, die bereits in anderen enthalten sind. Dazu direkt ein Beispiel. Betrachten wir Abb. 3.15.

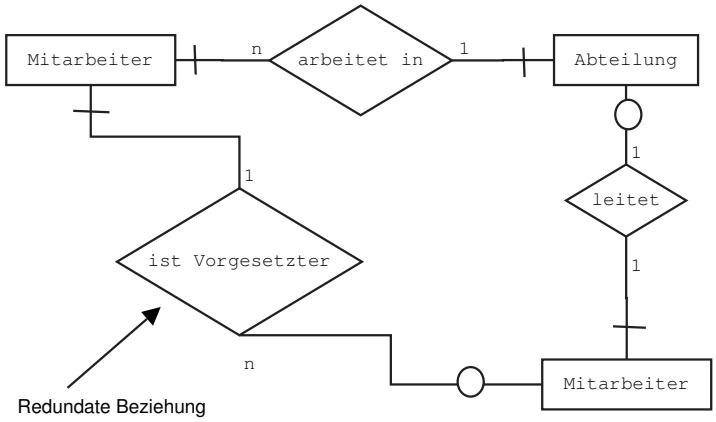


Abbildung 3.15: ERM: Redundante Beziehung

Zwei der dort aufgenommenen Beziehungen kennen Sie bereits, „Mitarbeiter arbeitet in Abteilung“ und „Mitarbeiter leitet Abteilung“. Neu aufgenommen habe ich die Symbole für Kann und Muss. Hinzugekommen ist die Beziehung „Mitarbeiter ist Vorgesetzter von Mitarbeiter“. Dies ist eine rekursive 1:n Beziehung, denn ein Mitarbeiter kann Vorgesetzter von mehreren Mitarbeitern sein, ein Mitarbeiter hat aber nur einen Vorgesetzten. Ein Mitarbeiter kann Vorgesetzter sein, muss aber nicht, andererseits muss ein Mitarbeiter einen Vorgesetzten haben. Klingt alles ganz gut, ist aber leider falsch. Die neue Beziehung ist nämlich überflüssig. Dadurch, dass ein Mitarbeiter in einer Abteilung arbeiten muss[4] und jede Abteilung einen Abteilungsleiter haben muss[5], liegt der Vorgesetzte ja bereits fest, es ist der Abteilungleiter.

Andererseits bedeutet dies nicht, dass solche Beziehungen immer redundant sind. Denn ändern wir z.B. die rekursive Beziehung zwischen Mitarbeiter und Mitarbeiter in „Mitarbeiter ist verheiratet mit Mitarbeiter“ (vgl. Abb. 3.16), so ist dies natürlich vollständig korrekt.

[4] Das bedeutet nämlich, wir können zu jedem Mitarbeiter seine Abteilung bestimmen.
[5] Das bedeutet nämlich, wir können zu jeder Abteilung den Abteilungsleiter bestimmen.

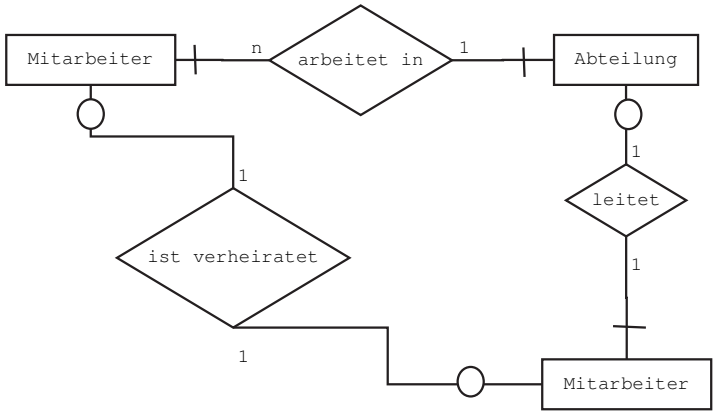


Abbildung 3.16: ERM: Nicht Redundante Beziehung

Auch eine kleine Änderung in Abb. 3.15 führt bereits zu einer nicht redundanten Beziehung. Gibt es zum Beispiel im Unternehmen Mitarbeiter, die keiner Abteilung angehören, dann wird aus einem Muss ein Kann und wir erhalten Abb. 3.17.

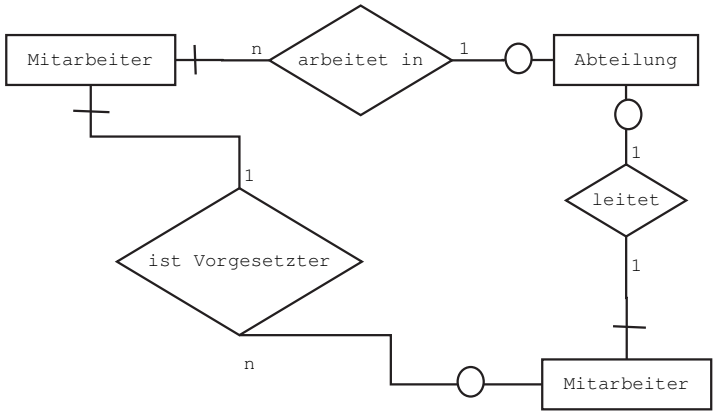


Abbildung 3.17: ERM: Nicht Redundante Beziehung 2

Nun ist die Beziehung „Mitarbeiter ist Vorgesetzter von Mitarbeiter“ nicht mehr redundant, denn wir können nicht mehr zu jedem Mitarbeiter über seinen Abteilungsleiter den Vorgesetzten ermitteln, weil ja Mitarbeiter ohne Abteilung existieren und damit auch ohne Abteilungsleiter und die Beziehung „Mitarbeiter ist Vorgesetzter von Mitarbeiter“ macht wieder Sinn und muss modelliert werden.

3.3 Vollständige ERM’s der drei Beispiele

Die vollständigen ERM’s der drei Beispiele können den Abbildungen 3.18, 3.19 und 3.20 entnommen werden.

3.4 Zusammenfassendes Beispiel

Zum Abschluss dieses Kapitels wollen wir ein ERM eines größeren Beispiels herleiten:
Promotordatenbank
Das Unternehmen abc-Promotion vermittelt Promotion-Kräfte für individuelle Verkaufs- und Werbeaktionen. Das kennt

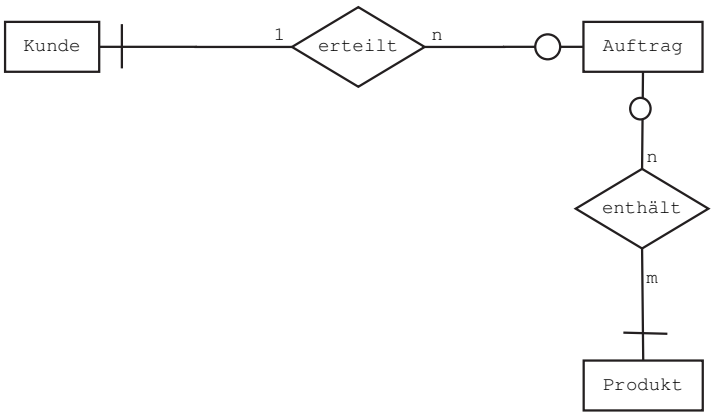


Abbildung 3.18: ERM Beispiel 3.1 Gesamtdarstellung

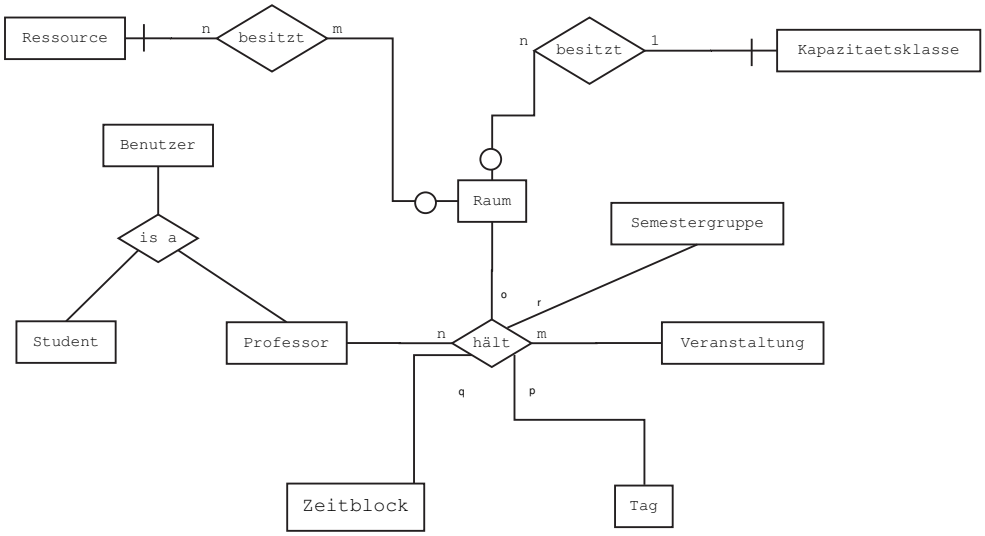


Abbildung 3.19: ERM Beispiel 3.2 Gesamtdarstellung

ein jeder von Ihnen: Sie betreten einen Supermarkt und in dem Supermarkt wird an einem Sonderstand ein Produkt eines bestimmten Herstellers besonders beworben.

Die Promotion-Aktionen sollen durch eine Datenbank unterstützt werden.

Als Promotoren werden Mitarbeiter des Unternehmens oder Selbstständige eingesetzt. Für alle Promotoren speichern wir Name, Vorname, email und Handy-Nummer, bei Selbstständigen zusätzlich Adresse und privates Telefon, bei Mitarbeitern zusätzlich die Durchwahl im Unternehmen und die Abteilung. Hierbei gilt: Ein Mitarbeiter gehört einer Abteilung an, einer Abteilung können natürlich mehrere Mitarbeiter angehören. Es gibt keine Abteilung ohne Mitarbeiter, Mitarbeiter, die keiner Abteilung angehören, haben wir in der Promotorendatenbank auch nicht.

Die Kunden sind die Hersteller der Konsumgüter. Hier speichern wir z.B. den Namen des Unternehmens, die Adresse und so weiter. Kunden sind Branchen zugeordnet. Bei den Branchen speichern wir den Namen der Branche und den WZ93Code. Der WZ93Code ist ein vom statistischen Bundesamt entwickelter nationaler Standardcode für eine Branche. Hierbei gilt: Ein Unternehmen kann mehreren Branchen zugeordnet sein, einer Branche gehören natürlich mehrere Unternehmen an. Es gibt keine Branche ohne Unternehmen, genauso wenig, wie es Unternehmen ohne Branche gibt.

Auch den Promotoren sind Branchen zugeordnet. Das liegt auf der Hand, denn jemand, der glaubwürdig Fiege-Pils bewerben kann, muss dazu für Champagner noch lange nicht in der Lage sein. Hier gilt: Ein Promoter ist mehreren Branchen zugeordnet, einer Branche können mehrere Promotoren zugeordnet sein. Es gibt keine Promotoren ohne Branche, genauso wenig, wie es Branchen ohne Promotoren gibt.

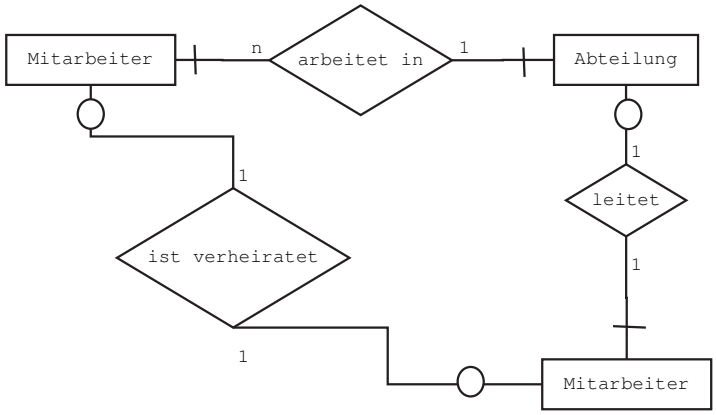


Abbildung 3.20: ERM: 3.3 Gesamtdarstellung

Promotoren werden darüber hinaus Supermärkten zugeordnet (das sind die Supermärkte, in denen sie eingesetzt werden können). Dabei gilt: Einem Supermarkt können mehrere Promotoren zugeordnet werden, ein Promotor kann mehreren Supermärkten zugeordnet werden. Es gibt keine Promotoren ohne Supermärkte, genauso wenig, wie es Supermärkte ohne Promotoren gibt. Für die Supermärkte speichern wir auch Namen, Adresse, etc.

Promotoren führen Promotion-Veranstaltungen in Supermärkten für Kunden an bestimmten Terminen durch. Dabei gilt: Ein Promotor kann mehrere Promotion-Veranstaltungen an unterschiedlichen Terminen für unterschiedliche Kunden in unterschiedlichen Supermärkten durchführen. Für einen Kunden können unterschiedliche Promotoren mehrere Promotion-Veranstaltungen an unterschiedlichen Terminen in unterschiedlichen Supermärkten durchführen.

3.4.1 Die Entities

Entities in unserem Beispiel sind auf jeden Fall:

1. Promotor
2. Mitarbeiter
3. Selbsständiger
4. Abteilung
5. Kunde
6. Branche
7. Supermarkt

Zusätzlich kann Termin noch als eigenständige Entity aufgenommen werden, muss aber nicht. Termin kann auch einfach als Attribut aufgefasst werden. Ich werde darauf näher in Kapitel 4 eingehen.

3.4.2 Die Beziehungen

Die Beziehungen in unserem Beispiel sind:

1. Eine Is-a-Beziehung zwischen Promotor, Selbstständiger und Mitarbeiter
2. Eine 1:n-Beziehung zwischen Mitarbeiter und Abteilung. Dies ist eine Muss Muss-Beziehung.
3. Eine n:m-Beziehung zwischen Kunde und Branche, die auch Muss Muss ist.
4. Eine n:m-Beziehung zwischen Promotor und Branche, die auch Muss Muss ist.

5. Eine n:m-Beziehung zwischen Promotor und Supermarkt, die ebenfalls Muss Muss ist.
6. Eine Beziehung zwischen mehr als 2 Entities:
    - Kunde
    - Supermarkt
    - Promotor
    - und ggf. Termin, falls Termin als eigenständige Entity modelliert wurde.

3.4.3 Das Modell

Abb. 3.21 zeigt das ERM als Grafik.

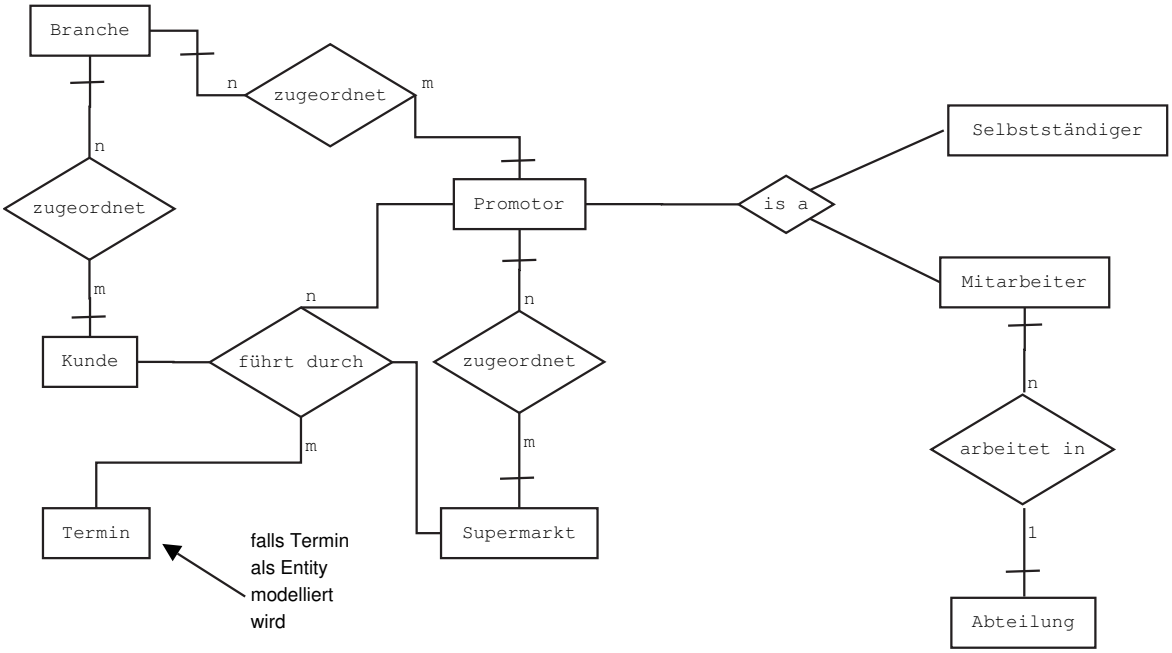


Abbildung 3.21: Das ERM des zusammenfassenden Beispiels

Kapitel 4

Vom ERM zur Tabellenstruktur

Nachdem wir aus einer gegebenen Problemstellung das ERM entwickeln können, bleibt noch, die Tabellenstruktur aus unserem Modell abzuleiten. Dies erfolgt, wie bereits in Kapitel 3 dargestellt, nach festgelegten Regeln.

4.1 Die Abbildung der Entities

Zunächst gilt:

Regel 4.1 *Jede Entity aus dem ERM führt zu einer Tabelle in der Datenbank.*

Hier überlegen wir uns jetzt auch die Attribute (Felder) der Tabellen. Die Attribute sind genau das, was wir zu den einzelnen Entities abspeichern wollen.
Alles weitere ergibt sich aus den Beziehungen, in denen die Entities zueinander stehen, und somit aus den Ergebnissen von Kapitel 3.2.

4.2 Die Abbildung der Beziehungen

4.2.1 1:n-Beziehung

Für 1:n-Beziehungen gibt es zwei Regeln, wie sie in die Tabellenstruktur überführt werden. Wir starten mit der einfacheren Regel[1].

Regel 4.2 *Wenn die 1-Entity einer 1:n-Beziehung nicht optional ist (also ein Muss-Teil), wird der Primärschlüssel der zur 1-Entity gehörigen Tabelle Fremdschlüssel in der zu n-Entity gehörenden Tabelle.*

Das klingt fürchterlich kompliziert, ist aber in Wahrheit und Wirklichkeit ganz einfach. Betrachten wir dazu noch einmal unser ERM zur Kunde - Auftrag Beziehung (noch einmal als Abb. 4.1 in den Text aufgenommen).

Abbildung 4.1: ERM: Kunde-Auftrag mit Kann Muss

Hier ist Kunde die 1-Entity (ein Auftrag kommt von einem Kunde). Darüber hinaus ist Kunde ein Muss-Teil (ein Auftrag muss schließlich von einem Kunden erteilt werden). Die Voraussetzung von Regel 4.2[2] ist somit erfüllt. Unsere Regel 4.2 besagt nun: Der Primärschlüssel der Tabelle der 1-Entity wird Fremdschlüssel in der Tabelle der n-Entity. Die 1-Entity ist Kunde, der Primärschlüssel der zugehörigen Tabelle ist KundeNr. KundeNr wird also Fremdschlüssel in Auftrag. Wir erhalten also die in Abb. 4.2 dargestellte Tabellenstruktur.

[1] Und aus Kapitel 1 wissen wir ja auch bereits, wie es geht.
[2] Für die Mathematik-Legastheniker: Voraussetzung ist das, was im Wenn-Teil des Satzes steht :-))).

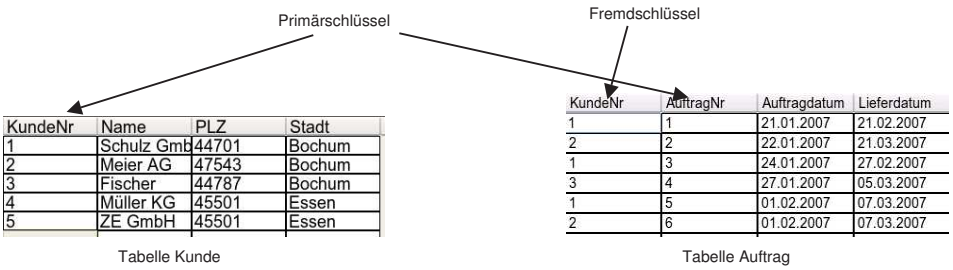
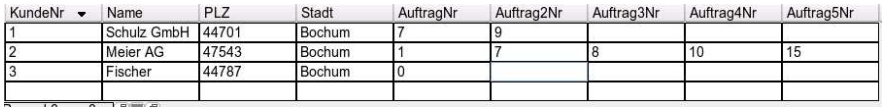


Abbildung 4.2: Korrekte Auflösung der 1:n-Beziehung Kunde Auftrag

Diese Auflösung einer 1:n-Beziehung ist auch logisch. Denn wie bereits in Kapitel 1 dargestellt, sind 1:n-Beziehung „anders herum“ nicht auflösbar. Eine AuftragNr in Kunde würde zu einer Tabelle in einer Tabelle führen. Damit ist gemeint, dass es dann in Kunde nicht nur ein Feld mit einer AuftragNr geben müßte, sondern mehrere, je nachdem, wie viele Aufträge der Kunde gerade offen hat. Das würde zu vielen nicht gefüllten Feldern in der Tabelle Kunde führen, nämlich bei all jenen Kundendatensätzen, wo gerade gar keine oder wenige Aufträge offen sind. Dies zeigt auch Abb. 4.3. Die Auftragnummer wurde in die Kundentabelle aufgenommen. Der Kunde mit KundeNr 2 hat zur Zeit fünf Aufträge offen, wir haben fünf Felder für Aufträge in der Kundentabelle. Kunde 3 hat zur Zeit keinen offenen Auftrag, was bedeutet, dass alle Felder für die Auftragnummern leer sind. Bei Kunde 1 sind noch zwei Aufträge offen, also sind drei AuftragNr-Felder leer. Diese Tabellenstruktur ist daher extrem ineffizient. Erschwerend kommt hinzu, dass, sobald ein Kunde über mehr als fünf offene Aufträge verfügt, die Struktur der Tabelle Kunde geändert werden muss. Es muss nämlich ein weiteres Feld Auftrag6Nr hinzugefügt werden.



KundeNr	Name	PLZ	Stadt	AuftragNr	Auftrag2Nr	Auftrag3Nr	Auftrag4Nr	Auftrag5Nr
1	Schulz GmbH	44701	Bochum	7	9			
2	Meier AG	47543	Bochum	1	7	8	10	15
3	Fischer	44787	Bochum	0				

Abbildung 4.3: Falsche Auflösung der 1:n-Beziehung

Zum Abschluss setzen wir die in Abb. 3.5 dargestellte 1:n-Beziehung zwischen Raum und Kapazitätsklasse in eine Tabellenstruktur um. Zunächst gibt es zwei Tabellen, die Tabelle Raum und die Tabelle Kapazitaetsklasse[3]. Nach Regel 4.2 wird nun der Primärschlüssel von Kapazitaetsklasse Fremdschlüssel in Raum und wir erreichen die in Abb. 4.4 dargestellte Tabellenstruktur.

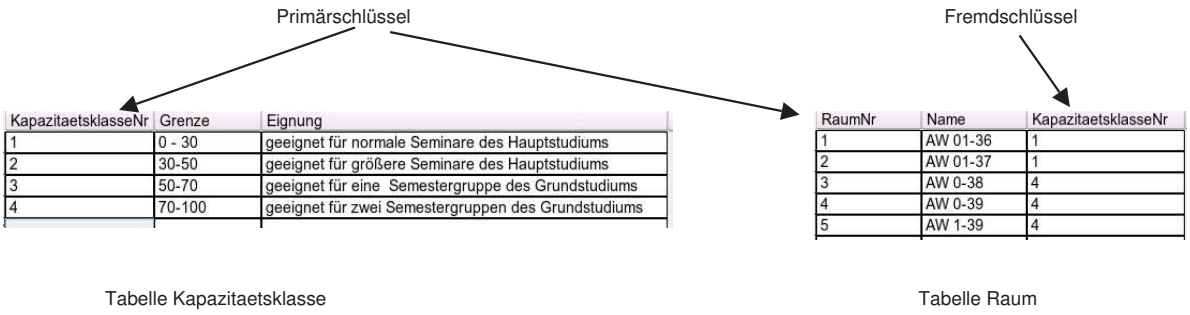


Abbildung 4.4: Auflösung der 1:n-Beziehung Raum Kapazitätsklasse

Bevor wir nun die zweite Regel zur Auflösung von 1:n-Beziehungen behandeln, betrachten wir n:m-Beziehungen[4].

[3] Bei Tabellennamen vermeide ich Umlaute.

[4] Das hat schon seinen tieferen Grund, denn, nachdem Sie verstanden haben, wie n:m-Beziehungen in Tabellen überführt werden, ist die zweite Regel für 1:n-Beziehungen trivial.

4.2.2 n:m-Beziehung

Wir beginnen mit einem Beispiel. Zur Abwechslung starten wir mit unserem Intranet-Informationssystem. Wir betrachten die n:m-Beziehung zwischen Raum und Ressource. Auch hier nehme ich der besseren Verständlichkeit wegen diese Beziehung als Abb. 4.5 erneut in den Text auf.

Abbildung 4.5: ERM: Raum-Ressource

Beginnen wir mit unseren Überlegungen. Klar ist (aus Regel 4.1), dass wir zumindest die beiden Tabellen Raum und Ressource in unserer Datenbank anlegen müssen. Den Primärschlüssel der Tabelle Raum nennen wir RaumNr, den von Ressource RessourceNr.

Eine Idee wäre, die RaumNr als Fremdschlüssel in Ressource zu übernehmen. Allerdings müßten Sie sofort erkennen können, dass diese Idee nicht zu den wirklich guten gehört. Denn wir bekommen bei dieser Vorgehensweise die selben Probleme, die wir haben, wenn wir eine 1:n-Beziehung falsch auflösen. Eine Ressource kann es in vielen Räumen geben, eine andere in nicht ganz so vielen. So haben z.B. die meisten unserer Vorlesungsräume eine normale Tafel[5]. Wir müßten also, da Tafel sicher eine Ressource ist, die wir in unserer Datenbank speichern müssen, so viele Felder für Räume in der Ressourcen-Tabelle aufnehmen, wie wir Räume mit Tafeln haben. Bei dem Datensatz der Ressource PC-Arbeitsplätze wäre das ziemlich übertrieben, denn wir haben z.Zt. nur zwei Räume, die mit PC-Arbeitsplätzen ausgestattet sind. Wir haben also die in Abb. 4.3 dargestellte Problematik, nur nicht mit Auftragsnummern in der Tabelle Kunde, sondern mit Raumnummern in der Tabelle Ressource[6].

Andererseits könnten wir ja die RessourceNr als Fremdschlüssel in Raum übernehmen. Dummerweise geht das auch nicht, weil das zur selben Problematik führt. Denn wenn Sie sich die Informatik-Übungsräume ansehen, die haben alles: Beamer, PC-Arbeitsplätze, Whiteboard, Overhead-Projektor, halt alles, was gut und teuer ist. Die Hörsäle, wo mein Kollege Wolik seine Veranstaltungen macht, sind nicht ganz so gut ausgestattet. Wir haben also die in Abb. 4.3 dargestellte Problematik, nur nicht mit Auftragsnummern in der Tabelle Kunde, sondern mit Ressourcenummern in der Tabelle Raum[7].

Verwunderlich ist das alles aber eigentlich gar nicht, denn es gibt hier keine 1-Entity, von der wir wissen, dass sie in dieser Beziehung nur einmal vorkommt[8]. Hier reichen die Tabellen, die wir aus Entities ableiten nicht mehr aus. Wir bekommen nun eine Tabelle aus der n:m-Beziehung hinzu. Die richtige Lösung zeigt Abb. 4.6.

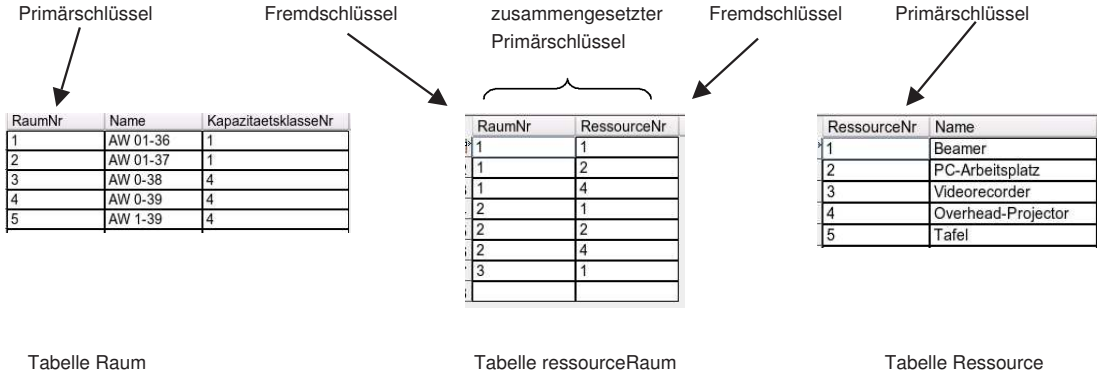


Abbildung 4.6: Auflösung der n:m-Beziehung Raum-Ressource

Aus einer n:m-Beziehung entsteht immer eine weitere Tabelle. Diese Tabelle nennen wir Verbindungs- oder Bezie-

[5] Bis auf ganz wenige, die ein Whiteboard haben.
[6] Ich verzichte hier auf eine erneute Abbildung, weil den Transfer bekommen Sie hin, oder ?? :-)))
[7] Ich verzichte hier erneut :-) auf eine erneute Abbildung, weil den Transfer bekommen Sie immer noch hin, oder ?? :-)))
[8] Wir sind ja gerade bei n:m-Beziehungen.

hungstabelle. Verbindungstabellen besitzen immer einen zusammengesetzten Primärschlüssel. Dies können wir auch Abb. 4.6 entnehmen. Die Verbindungstabelle, die aus der n:m-Beziehung ensteht, bekam den Namen ressourceRaum. RaumNr ist kein Primärschlüssel dieser Tabelle, denn es gibt mehrere Datensätze mit der gleichen RaumNr (z.B. Datensätze 1 und 2 mit der RaumNr 1). RessourceNr ist aber auch nicht Primärschlüssel der Tabelle ressourceRaum, denn es gibt mehrere Datensätze mit der gleichen RessourceNr (z.B. Datensätze 1 und 4 mit der RessourceNr 1). Mehr Felder haben wir nicht in der Tabelle, also kann ein einzelnes Feld nicht Primärschlüssel dieser Tabelle sein. Betrachten wir aber RessourceNr und RaumNr zusammen. Es gibt keine zwei Datensätze in der Tabelle ressourceRaum mit gleichen Werten für RaumNr und RessourceNr. Das ist aber genau die Definition eines Primärschlüssels. RaumNr und RessourceNr zusammen sind also Primärschlüssel der Verbindungstabelle. RaumNr alleine ist Fremdschlüssel zur Tabelle Raum (RaumNr ist ja Primärschlüssel dieser Tabelle). RessourceNr alleine ist Fremdschlüssel zur Tabelle Ressource (RessourceNr ist ja Primärschlüssel dieser Tabelle). Die Verbindungstabelle besitzt keine weiteren Attribute. Dies muss nicht so sein, wir werden auch Beispiele mit weiteren Attributen in der Verbindungstabelle kennen lernen. Allerdings müssen wir hier vorsichtig sein. Weitere Attribute einer Verbindungstabelle müssen vom zusammengesetzten Primärschlüssel insgesamt abhängig sein. Denn sonst würden diese Attribute zu einer der Grundtabellen gehören. Wir werden diesen Sachverhalt im Folgenden aber eingehend betrachten. Auch der zuammengesetzte 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. Auch solche Beispiele werden wir noch kennen lernen.
Allgemein können wir sagen:

Regel 4.3 *Bei der Auflösung einer n:m-Beziehung ensteht aus dieser Beziehung eine weitere Tabelle. Diese Tabelle besitzt einen zusammengesetzten Primärschlüssel. Bestandteile des zusammengesetzten Primärschlüssel sind die Primärschlüssel der Grundtabellen der n:m-Beziehung. Der Primärschlüssel der Verbindungstabelle kann weitere Attribute enthalten. Die Tabelle selber kann ebenfalls weitere Attribute enthalten, diese müssen aber vom Primärschlüssel insgesamt abhängig sein.*

Wir sehen hier, dass die Abbildung einer n:m-Beziehung in Tabellen unabhängig von der in Kapitel 3.2.6 dargestellten Kann-Muss-Eigenschaft von Beziehungen ist. Wir ermitteln diese für n:m-Beziehungen aber trotzdem und zwar aus zwei Gründen:

1. Wir bekommen so ein besseres Verständnis der Problemstellung.
2. Zur Ermittlung von Redundanzen (vgl. Kapitel 3.2.8).

Als zweites Beispiel betrachten wir die n:m-Beziehung zwischen Produkt und Auftrag aus Abb. 3.4. Eine mögliche Tabellenstruktur zu dieser Problemstellung zeigt Abb. 4.7.

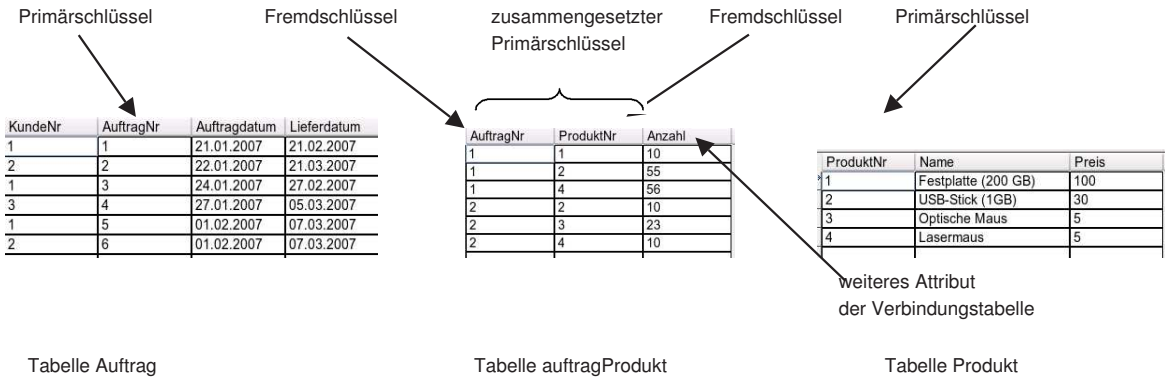


Abbildung 4.7: Auflösung der n:m-Beziehung Auftrag-Produkt

Beachten Sie zunächst die kluge Auswahl von Auftrag- und Lieferdatum. Professoren sind schon clever. So gelingt es nämlich, ein Script länger aktuell zu halten :-). Aber Spaß beiseite. Hier haben wir, wie oben bereits angekündigt, ein weiteres Attribut in der Verbindungstabelle. Es ist das Attribut „Anzahl“. Dieses Attribut können wir in keiner anderen Tabelle speichern. Wir können „Anzahl“ nicht in Auftrag aufnehmen, denn die Anzahl der Produkte in einem Auftrag ist

nicht nur vom Auftrag abhängig, sondern auch vom bestellten Produkt. Wir können „Anzahl“ aber auch nicht in Produkt aufnehmen, denn die Anzahl eines Produkts in einem Auftrag ist nicht nur vom Produkt abhängig, sondern auch vom Auftrag. Wir stellen also fest, die Anzahl eines Produktes in einem Auftrag ist von beidem abhängig von Produkt und Auftrag gemeinsam. Und solche Attribute gehören in die Verbindungstabelle.

Die in Abb. 4.7 gezeigte Tabellenstruktur setzt voraus, das Aufträge immer als Einheit geliefert werden. Teillieferungen sind nicht zulässig. Sie erkennen dies daran, dass das Feld Lieferdatum ein Feld der Tabelle Auftrag ist. Dies bedeutet, wir können pro Auftrag genau ein Lieferdatum speichern, und dies bedeutet damit natürlich auch, dass Teillieferungen nicht möglich sind. Wollen wir Teillieferungen im Datenmodell abbilden, so müssen wir das Feld Lieferdatum in der Tabelle Auftrag streichen und in die Tabelle auftragProdukt aufnehmen (vgl. 4.8) . Man sieht daran sehr schön, dass der Entwurf einer Datenstruktur immer von den abzubildenden Prozessen abhängt.

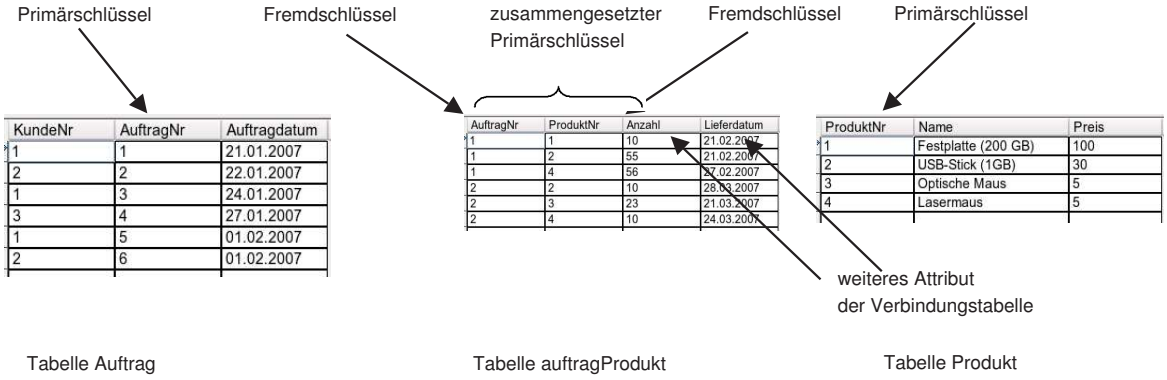


Abbildung 4.8: Alternative Auflösung der n:m-Beziehung Auftrag-Produkt

Aber selbst, wenn wir das Lieferdatum in auftragProdukt verschieben, sind wir nicht ganz flexibel. Die neue Struktur erlaubt keine beliebigen Teillieferungen, sondern nur, dass wir die gesamte Menge eines Produkts in einem Auftrag teilliefern. Wollen wir vollständige Flexibilität, reicht unsere Tabellenstruktur nicht mehr aus. Wir haben dann eine Beziehung zwischen Auftrag und Produkt nicht modelliert, nämlich „Produkt wurde geliefert aus Auftrag“. Dies ist eine n:m-Beziehung. Aus einem Auftrag müssen[9] mehrere Produkte teilgeliefert werden, ein Produkt kann für mehrere Aufträge teilgeliefert werden. Sie ist in Abb. 4.9 dargestellt.

Abbildung 4.9: ERM: Produkt-Auftrag mit Teillieferung

Abb. 4.10 zeigt die sich nun ergebende Tabellenstruktur. Das Feld Lieferdatum verschwindet aus auftragProdukt. Wir erhalten aus der zweiten n:m-Beziehung zwischen Auftrag und Produkt eine neue Verbindungstabelle, die Tabelle Lieferung. Hier tritt jetzt auch der Fall ein, dass wir den Primärschlüssel der Verbindungstabelle erweitern müssen. Denn ProduktNr und AuftragNr reichen jetzt zur Identifizierung eines Datensatzes nicht mehr aus. U.a. kann man das an den Datensätzen 1 und 2 der Verbindungstabelle erkennen. Hier stimmen ProduktNr und AuftragNr überein (beides 1). Die beiden Datensätze unterscheiden sich allein durch ihr Lieferdatum. Dies bedeutet, dass das Lieferdatum Teil des Primärschlüssels werden muss[10].

[9] Kein Auftrag ohne Lieferung.

[10] Eine Alternative ist, das Lieferdatum als eigene Entity zu modellieren. Aus einer n:m-Beziehung würde dann eine Beziehung zwischen mehr als zwei Entities.

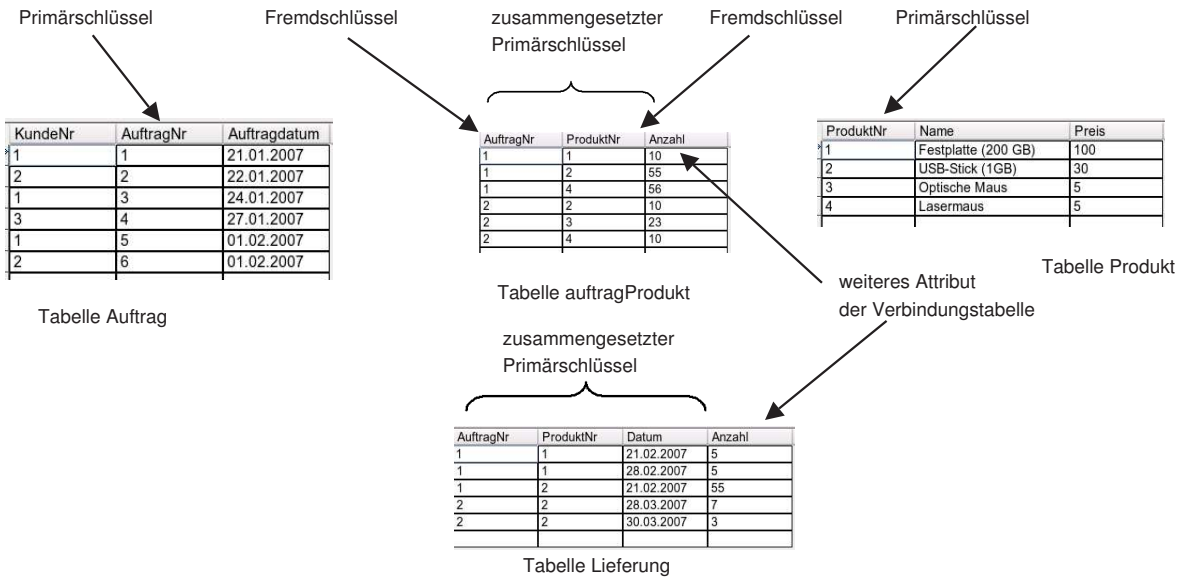


Abbildung 4.10: Auflösung der n:m-Beziehung Auftrag-Produkt mit Teillieferung

4.2.3 1:n-Beziehung, zum Zweiten

Regel 4.4 *Wenn die n-Entity einer 1:n-Beziehung optional ist (also ein Kann-Teil) und die Beziehung selten eintritt, entsteht, wie im Falle der n:m-Beziehung, aus dieser Beziehung eine weitere Tabelle. Diese Tabelle besitzt einen zusammengesetzten Primärschlüssel. Bestandteile des zusammengesetzten Primärschlüssel sind die Primärschlüssel der Grundtabellen der n:m-Beziehung.*

Zur Veranschaulichung betrachten wir wieder unsere Kunden-Auftrag-Beziehung. Wir ändern die Aufgabenstellung folgendermaßen: Wir haben nicht zu allen Aufträgen Kunden, wir produzieren normalerweise z.B. für das Lager. Ganz selten führen wir einen Großauftrag für einen Kunden durch (dies ist mit „die Beziehung selten eintritt“ in Regel 4.4 gemeint). Die neue Beziehung ist in Abb. 4.11 dargestellt.

Abbildung 4.11: ERM: Aufträge ohne Kunden

Die in Abb. 4.2 dargestellte Lösung hat nun den Nachteil, das das Feld KundeNr in Auftrag (der Fremdschlüssel) häufig leer bleibt. In einem solchen Fall, der allerdings in der Modellierung nicht eben häufig vorkommt, behandeln wir eine 1:n-Beziehung genauso, wie eine n:m-Beziehung. Wir erzeugen eine Verbindungstabelle. Auf eine Abbildung und eine weitere Diskussion dieses Falls verzichte ich, der Stoff ist ja in Kapitel 4.2.2 ausführlich dargestellt.

4.2.4 1:1-Beziehung

Bei 1:1-Beziehungen gibt es, in Abhängigkeit vom Ergebnis der Kann-Muss-Analyse, drei Regeln zur Überführung der Beziehung in die Tabellenstruktur.

Muss-Muss

Regel 4.5 *Wenn beide Entities einer 1:1-Beziehung nicht optional sind (also Muss-Teile), wird die 1:1-Beziehung in einer Tabelle abgebildet. Die zu den Entities gehörenden Grundtabellen werden also zu einer zusammengefasst. Der*

Primärschlüssel der neuen Tabelle ist der Primärschlüssel einer der Grundtabellen 1:1-Beziehung. Jeder von beiden kann genommen werden.

Solche 1:1-Beziehungen kommen in der Praxis auch nicht häufig vor. Zudem ist eine solche Beziehung ein Indiz für einen Fehler in der Modellierung. Viel deutet nämlich darauf hin, dass es sich in so einem Fall nicht um zwei, sondern in Wirklichkeit nur um eine Entity handelt.

Kann-Muss

Regel 4.6 *Wenn eine Entity einer 1:1-Beziehung nicht optional (also Muss-Teil), die andere hingegen optional (also Kann-Teil) ist, wird der Primärschlüssel der zur Muss-Entity gehörenden Tabelle als Fremdschlüssel in die zur Kann-Entity gehörende Tabelle übernommen.*

Wir machen uns diesen Sachverhalt am Beispiel der Beziehung Mitarbeiter-Abteilung (vgl. Abb. 3.8) klar. Dies ist ja eine 1:1 Kann-Muss-Beziehung. Mitarbeiter ist der Muss-Teil. Die Abbildung der Mitarbeiter-Abteilung-Beziehung zeigt Abb. 4.12.

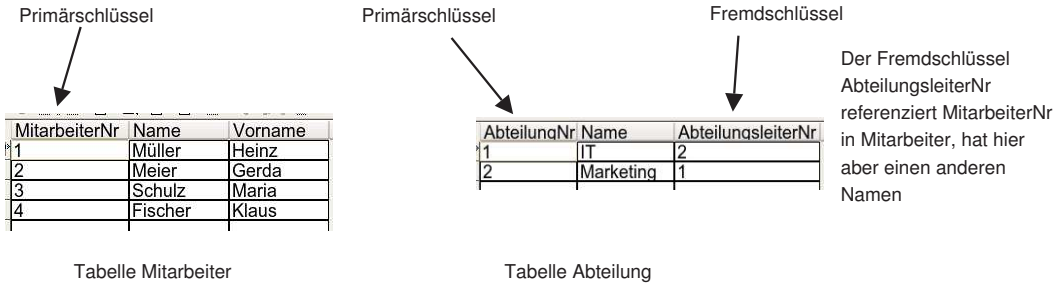


Abbildung 4.12: Auflösung der 1:1-Beziehung Mitarbeiter-Abteilung

Der Primärschlüssel von Mitarbeiter wird in Abb. 4.12 wie in Regel 4.6 in die Tabelle Abteilung als Fremdschlüssel übernommen. Allerdings, und das ist auch neu für Sie, erhält die MitarbeiterNr in der Tabelle Abteilung einen anderen Namen, nämlich AbteilungsleiterNr. Das ist aber für Datenbanksysteme kein Problem. Man muss „nur“ der Datenbank „klar machen“, welches Feld (welcher Primärschlüssel) durch den Fremdschlüssel AbteilungsleiterNr referenziert wird. Wie man das mit SQL erreicht, lernen Sie in Kapitel 5.3, die Vorgehensweise bei grafischen Schnittstellen zu Datenbanken, wie Access oder Rekall, lernen Sie in den Übungen.

Bie den in Abb. 4.12 dargestellten Daten ist also Frau Gerda Meier (AbteilungsleiterNr=2, daher wird der Datensatz mit der MitarbeiterNr 2 in Mitarbeiter selektiert) Leiterin der IT-Abteilung, wohingegen Herr Heinz Müller (AbteilungsleiterNr=1, daher wird der Datensatz mit der MitarbeiterNr 1 in Mitarbeiter selektiert) der Marketing-Abteilung vorsteht.

Kann-Kann

Regel 4.7 *Wenn beide Entities einer 1:1-Beziehung optional (also Kann-Teile) sind und die Beziehung selten eintritt, entsteht, wie im Falle der n:m-Beziehung, aus dieser Beziehung eine weitere Tabelle. Der Primärschlüssel der neuen Tabelle ist der Primärschlüssel einer der Grundtabellen der 1:1-Beziehung. Jeder von beiden kann genommen werden. Tritt die Beziehung jedoch häufig ein, wird wie in Regel 4.6 verfahren, wobei es egal ist, welcher Tabelle der Primärschlüssel der anderen als Fremdschlüssel hinzugefügt wird.*

Auf eine weitere Diskussion dieses Falls verzichte ich, der Stoff ist ja in Kapitel 4.2.2 ausführlich dargestellt.

4.2.5 Beziehungen zwischen mehr als 2 Entities

Beziehungen zwischen mehr als 2 Entities werden wie n:m-Beziehungen behandelt.

Regel 4.8 *Bei der Auflösung einer Beziehung zwischen mehr als 2 Entities ensteht aus der dieser Beziehung eine weitere Tabelle. Diese Tabelle besitzt einen zusammengesetzten Primärschlüssel. Bestandteile des zusammengesetzten Primärschlüssel*

sind die Primärschlüssel der Grundtabellen der Beziehung. Der Primärschlüssel der Verbindungstabelle kann weitere Attribute enthalten. Die Tabelle selber kann ebenfalls weitere Attribute enthalten, diese müssen aber vom Primärschlüssel insgesamt abhängig sein.

Abb. 4.13 zeigt die Auflösung der Stundenplan-Beziehung in Tabellen. Die in Abb. 4.13 dargestellten Screenshots stammen aus dem echten Intranet-System des Fachbereichs[11]. Im Intranet benutzen wir aber englische Bezeichnungen, so dass die Tabellennamen in Abb. 4.13 und die Namen der Entities in Abb. 3.10 nicht übereinstimmen[12].

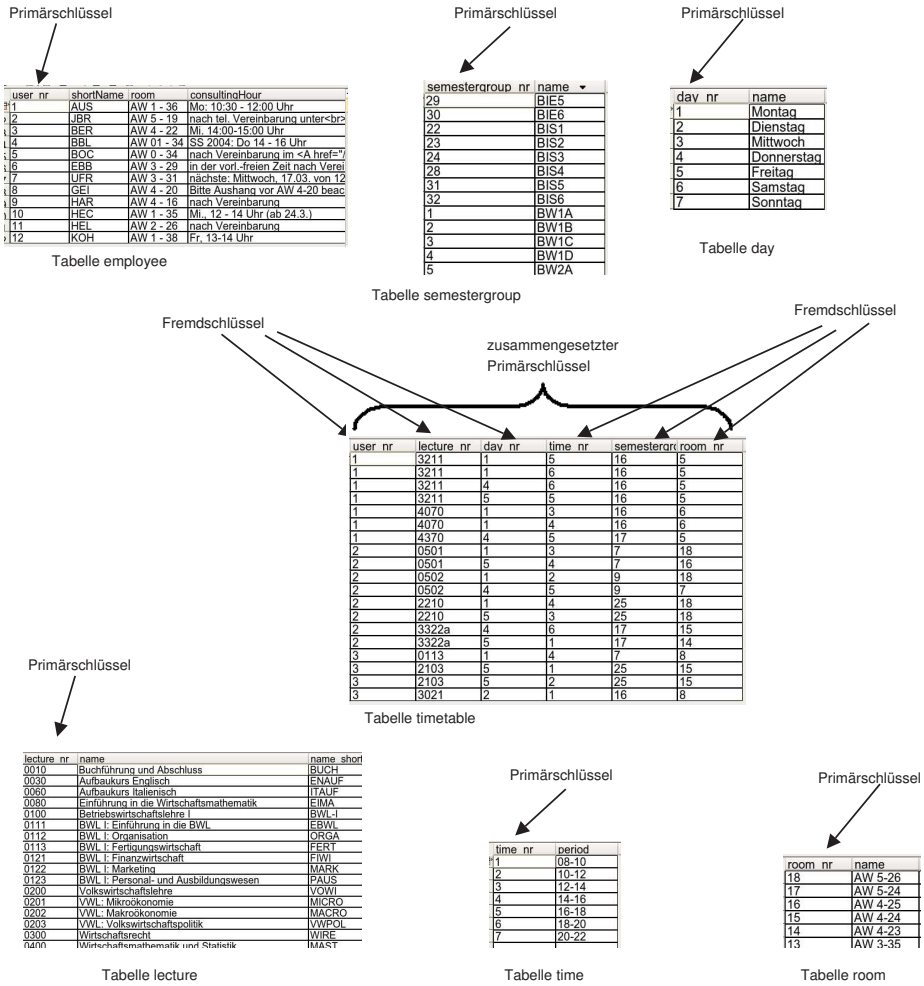


Abbildung 4.13: Auflösung Stundenplan-Beziehung

4.2.6 Generalisierung - Spezialisierung, Is-a-Beziehung

Zum Abschluss bleibt noch die Abbildung der Is-a-Beziehung in die Tabellenstruktur. Das ist aber glücklicherweise sehr einfach.

Regel 4.9 *Bei der Auflösung einer is-a-Beziehung werden die Attribute, die allen Spezialisierungen gemeinsam sind in die Tabelle der Generalisierungs-Entity aufgenomen. Die Attribute, die die Spezialisierungen kennzeichnen in ihren jeweiligen Tabellen. Der Primärschlüssel aller drei Tabellen ist der Primärschlüssel der Tabelle der Generalisierungs-Entity.*

Abb. 4.14 zeigt die Auflösung der is-a-Beziehung Benutzer, Professor, Student unseres Intranet-Anwendung. Auch hier verwende ich wieder die englischen Bezeichnungen der Original-Tabellen.

[11] Ich bin ja auch faul, und so mußte ich keine Daten eingeben.

[12] Es ist nämlich die englische Übersetzung, aber das sollten Sie hinbekommen, denke ich.

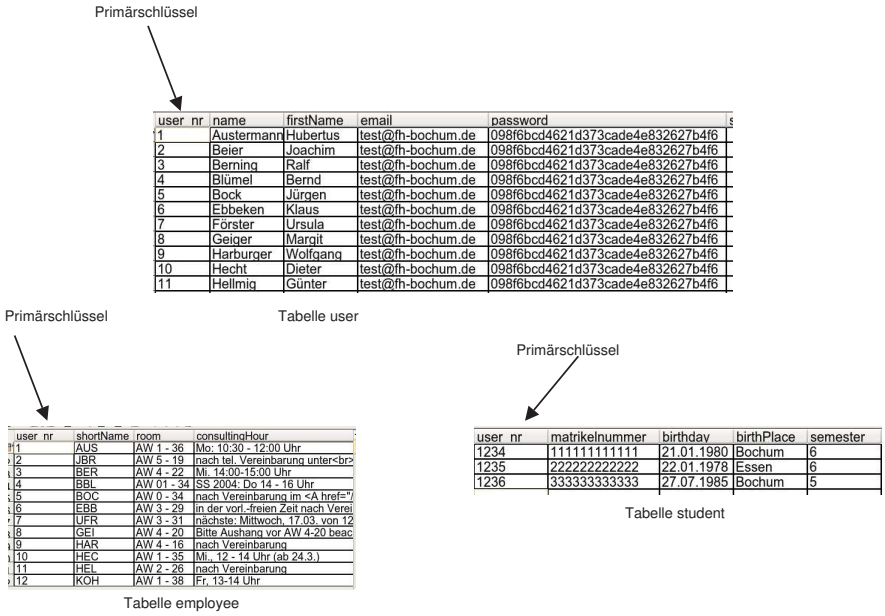


Abbildung 4.14: Auflösung Beziehung Benutzer-Student-Professor

4.2.7 Zusammenfassung: Die Tabellenstrukturen der Beispiele

Ich stelle aus Gründen der Übersichtlichkeit in diesem Kapitel die vollständigen Tabellenstrukturen der drei Beispiele dar. Primärschlüssel sind unterstrichen, Fremdschlüssel kursiv dargestellt.

Die Tabellenstruktur von Beispiel 3.1 bzw. Abb. 3.18

Name	Primärschlüssel	Weitere Felder
Kunde	KundeNr	Name, PLZ, Stadt
Auftrag	AuftragNr	AuftragDatum, Lieferdatum, *KundeNr*
Produkt	ProduktNr	Name, Preis
auftragProdukt	*AuftragNr*, *ProduktNr*	Anzahl

Die Tabellen zu 3.1

Die Tabellenstruktur von Beispiel 3.1 bzw. Abb. 3.18, mit Teillieferung gemäß Abb. 4.9

Name	Primärschlüssel	Weitere Felder
Kunde	KundeNr	Name, PLZ, Stadt
Auftrag	AuftragNr	AuftragDatum, *KundeNr*
Produkt	ProduktNr	Name, Preis
auftragProdukt	*AuftragNr*, *ProduktNr*	Anzahl, Lieferdatum

Die Tabellen zu Beispiel 3.1 mit Teillieferung

Die Tabellenstruktur von Beispiel 3.1 bzw. Abb. 3.18, mit flexibler Teillieferung gemäß Abb. 4.11

Name	Primärschlüssel	Weitere Felder
Kunde	KundeNr	Name, PLZ, Stadt
Auftrag	AuftragNr	AuftragDatum, *KundeNr*
Produkt	ProduktNr	Name, Preis
auftragProdukt	*AuftragNr*, *ProduktNr*	Anzahl
Lieferung	*AuftragNr*, *ProduktNr*, Lieferdatum	Anzahl

Die Tabellen zu Beispiel 3.1 mit flexibler Teillieferung

Die Tabellenstruktur von Beispiel 3.2 bzw. Abb. 3.19

Da ich hier wieder die Originaltabellen aus dem Intranet benutze, entsprechen die Bezeichnungen nicht den Entities in Abb. 3.19, sondern sind ihre englischen Übersetzungen.

Name	Primärschlüssel	Weitere Felder
user (Benutzer)	user_nr	name, firstName, email, password, phone, phoneMobile, ...
employee (Mitarbeiter, Professor)	user_nr	shortName, room, consultingHour, url, ...
student (Student :-))	user_nr	matrikelnummer, birthday, birthPlace, semester, ...
semestergroup (Semestergruppe)	semestergroup_nr	name
day (Tag)	day_nr	name
time (Zeit)	time_nr	period
room (Raum)	room_nr	name
lecture (Veranstaltung)	lecture_nr	name, name_short, name_en, sws, ects, ...
timetable (Stundenplan)	*lecture_nr*, *user_nr*, *semestergroup_nr*, *day_nr*, *time_nr*, *room_nr*	keine

Die Tabellen zu Beispiel 3.2

Die Tabellenstruktur von Beispiel 3.3 bzw. Abb. 3.20

Name	Primärschlüssel	Weitere Felder
Mitarbeiter	MitarbeiterNr	Name, Vorname, ...
Abteilung	AbteilungNr	Name, *AbteilungsleiterNr*
mitarbeiterHeirat	*MitarbeiterNrFrau*, *MitarbeiterNrMann*	keine

Die Tabellen zu Beispiel 3.3

4.2.8 Die Tabellenstrukturen des Beispiels aus Kapitel 3.4

In diesem Beispiel haben wir sieben (bzw. acht, falls Termin als eigenständige Entity modelliert wurde) Entities, daraus ergeben sich ebenfalls sieben (bzw. acht Grundtabellen). Es gibt eine is-a-Beziehung. Hier müssen wir nur die Attribute korrekt zwischen den Tabellen aufteilen und den Primärschlüssel der Generalisierungstabelle (Promotor) in die Spezialisierungen (Mitarbeiter, Selbstständiger) übernehmen.

Es existiert eine 1:n-Beziehung (Mitarbeiter - Abteilung), die wir aber bereits kennen.

Wir haben drei n:m-Beziehungen, erhalten also daraus auch 3 Verbindungstabellen. Darüber hinaus gibt es eine Beziehung zwischen mehr als zwei Entities, auch hier erhalten wir eine Verbindungstabelle.

Name	Primärschlüssel	Weitere Felder
Promotor	PromotorNr	Name, Vorname, email, telMobile
Mitarbeiter	PromotorNr	Durchwahl, *AbteilungNr*
Selbststaendiger	PromotorNr	PLZ, Stadt, Strasse, Hausnummer, telprivat
Abteilung	AbteilungNr	Name
Kunde	KundeNr	Name, PLZ, Stadt, Strasse, Hausnummer, ...
Branche	WZ93Code	Name
Supermarkt	SupermarktNr	PLZ, Stadt, Strasse, Hausnummer
Termin	TerminNr	Datum
brancheKunde	*KundeNr*, *WZ93Code*	keine
branchePromotor	*PromotorNr*, *WZ93Code*	keine
promotorSupermarktHeirat	*PromotorNr*, *SupermarktNr*	keine
Promotionsveranstaltung	*PromotorNr*, *SupermarktNr*, *KundeNr*, *TerminNr*	keine

Die Tabellen zum Beispiel in Kapitel 3.4

Modellieren wir Termin nicht als eigene Entity, fällt die Tabelle Termin weg und in die Tabelle Promotionsveranstaltung wird das Datum der Promotionsveranstaltung anstelle der TerminNr Teil des zusammengesetzten Primärschlüssels.

Kapitel 5

SQL

SQL (Structured Query Language) ist eine standardisierte Datenbanksprache. Durch SQL-Befehle kann man:

- Datenbanken anlegen.
- Tabellen anlegen.
- Datensätze in Tabellen eintragen.
- Datensätze aus Tabellen löschen.
- Datensätze in Tabellen ändern.
- Die in der Datenbank enthaltenen Informationen in beliebigen sinnvollen[1] Kombinationen aus der Datenbank extrahieren.

In einigen Anwendungsbereichen sind SQL-Kenntnisse nicht notwendig. Wenn Sie z.B. ausschließlich mit MS-Access, Rekall oder vergleichbaren Programmen arbeiten, benötigen Sie im Normalfall keine SQL-Kenntnisse. Denn wie Sie ja in der Übung bereits gelernt haben, sind Access-Datenbanken einfach Dateien des Windows-Dateisystems. Eine Datenbank anlegen ist also einfach eine Datei anlegen. Und auch die anderen, oben dargestellten Anwendungen, unterstützt Access durch grafische Werkzeuge.

Dies findet aber Grenzen. Abfragen auf eine Datenbank können so komplex werden, dass Sie sie mit grafischen Werkzeugen nicht mehr erstellen können[2]. Aber es kann der Fall eintreten, dass Sie aus einem Programm heraus in einer Datenbank gespeicherte Informationen verarbeiten wollen, z.B. um Unternehmenskennzahlen oder Umsatzverläufe oder Ähnliches in einer Tabellenkalkulation darzustellen. Dann müssen Sie aus Ihrem Programm heraus SQL-Abfragen an die Datenbank schicken. Oder aber Sie erhalten kein grafisches Frontend zur Datenbank, die Sie benutzen müssen. Oder durch das grafisches Frontend werden die Abfragen zu langsam. In all diesen Fällen ist es günstig, ein wenig SQL zu können. Glücklicherweise ist SQL keine richtige Programmiersprache[3], sondern richtig einfach.

Alle SQL-Schlüsselworte (wie select, where, etc.), die sie in den folgenden Kapiteln kennen lernen werden, sind nicht „case-sensitive“. Das bedeutet, Groß- Kleinschreibung spielt für SQL-Schlüsselworte keine Rolle (select = Select = SELECT = seleCT). Bei Tabellen- und Feldnamen kann das vom Datenbanksystem abhängig sein. Daher halten wir uns hier an die Groß- Kleinschreibung der Tabellen- und Feldnamen in der Datenbank.

Als Beispiel benutzen wir die in Kapitel 4.2.7 dargestellte Tabellenstruktur des ERM’s aus Abb. 3.18. Ich füge sie zur besseren Übersichtlickeit noch einmal ein:

Name	Primärschlüssel	Weitere Felder
Kunde	KundeNr	Name, PLZ, Stadt
Auftrag	AuftragNr	AuftragDatum, Lieferdatum, *KundeNr*
Produkt	ProduktNr	Name, Preis
auftragProdukt	*AuftragNr*, *ProduktNr*	Anzahl

Die für SQL als Beispiel genutzten Tabellen

[1] Und natürlich auch sinnfreien ...

[2] Solche werden wir allerdings im ersten Semester auch nicht behandeln :-).

[3] Programmieren lernen Sie, wenn wir mit VBA anfangen.

5.1 Einfache Abfragen auf eine Tabelle

Die einfachste SQL-Abfrage hat die Form:

SQL 5.1 *SQL mit „Sternchen“*

```
Select *
from Kunde
```

Ausgegeben werden alle Datensätze der Tabelle Kunde und zwar alle Felder eines jeden Datensatzes.
Wir können die Felder, die in der Ausgabe erscheinen, einschränken.

SQL 5.2 *SQL mit Attribut-Einschränkung*

```
Select Name, Stadt
from Kunde
```

Die Datensätze der Ergebnismenge können sortiert werden.

SQL 5.3 *SQL mit Sortierung*

```
Select Name, Stadt
from Kunde
order by PLZ, Name
```

Die Sortierung kann gedreht werden.

SQL 5.4 *SQL mit reversiver Sortierung*

```
Select Name, Stadt
from Kunde
order by PLZ desc, Name desc
```

Wir können nur unterschiedliche Werte ausgeben lassen.

SQL 5.5 *SQL mit „distinct“*

```
Select distinct PLZ
from Kunde
```

Existieren in unserer Tabelle Kunde mehrere Datensätze mit gleicher PLZ, so erscheint diese PLZ in der Ausgabe dennoch nur einmal (vgl. Abb. 5.1).

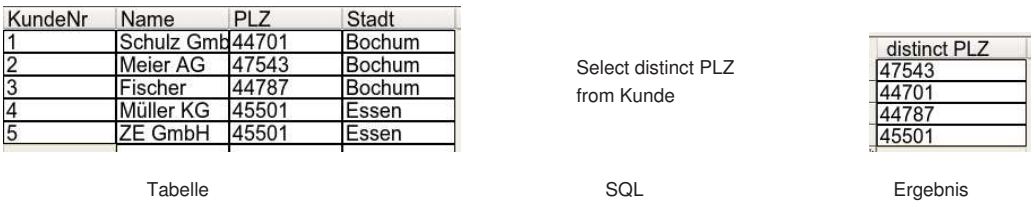


Abbildung 5.1: Ausgabe von SQL 5.5

Die Ausgabe kann eingeschränkt werden. Ausgegeben werden alle Bochumer Kunden. Beachten Sie, das der Wert gegen den verglichen wird, sofern er keine Zahl ist, in Apostrophe oder Anführungszeichen gesetzt werden muss.

SQL 5.6 *SQL mit Bedingung, Textfeld in der Bedingung*

```
Select Name, Stadt
from Kunde
where Stadt='Bochum'
order by PLZ desc, Name desc
```

Zahlen werden nicht in in Apostrophe oder Anführungszeichen gesetzt.

SQL 5.7 *SQL mit Bedingung, Zahlfeld in der Bedingung*

```
Select Name, Stadt
from Kunde
where KundeNr=1
```

Bedingungen werden durch and verknüpft.

SQL 5.8 *SQL mit mehreren Bedingungen*

```
Select Name, Stadt
from Kunde
where Stadt='Bochum'
and Name='Meier AG'
```

Innerhalb der Bedingungen können folgende Operatoren genutzt werden: $=, <, <=, >, >=, <>$ $(ungleich)$. Bedingungen können, wie in SQL 5.8 bereits gezeigt, durch logische Ausdrücke verknüpft werden. Neben „and“ sind noch „or“ und „not“ wichtig.

SQL 5.9 *SQL mit not*

```
Select Name, Stadt
from Kunde
where not (Stadt='Bochum')
```

Beachten Sie die Klammersetzug in SQL 5.9. Wie in der Mathematik (Punktrechnung vor Strichrechnung im einfachsten Fall) ist die Reihenfolge, in der durch logische Operatoren verbundene Bedingungen ausgewertet werden, festgelegt. Selber können Sie durch Klammerung andere Reihenfolge festlegen. Wenn Sie nicht genau wissen, wie die Datenbank sich verhalten wird, ist das Setzen von Klammern immer eine gute Idee. Dazu ein Beispiel:

SQL 5.10 *SQL mit Klammerung (1)*

```
select Name
from Kunde
where KundeNr=1
and (Stadt='Bochum' or Stadt='Essen')
```

SQL 5.11 *SQL mit Klammerung (2)*

```
select Name
from Kunde
where (KundeNr=1 and Stadt='Bochum')
or Stadt='Essen'
```

Das Ergebnis ist in Abb. 5.2 dargestellt.

SQL kann rechnen und zwar sowohl im Select, als auch in den Bedingungen. Dies zeigen SQL 5.12 und Abb. 5.3. Hier sieht man auch, das man die Spaltenüberschriften der Ergebnisse eines SQL-Kommandos ändern kann. Dies geschieht über das SQL-Schlüsselwort As.

SQL 5.12 *SQL mit Berechnungen*

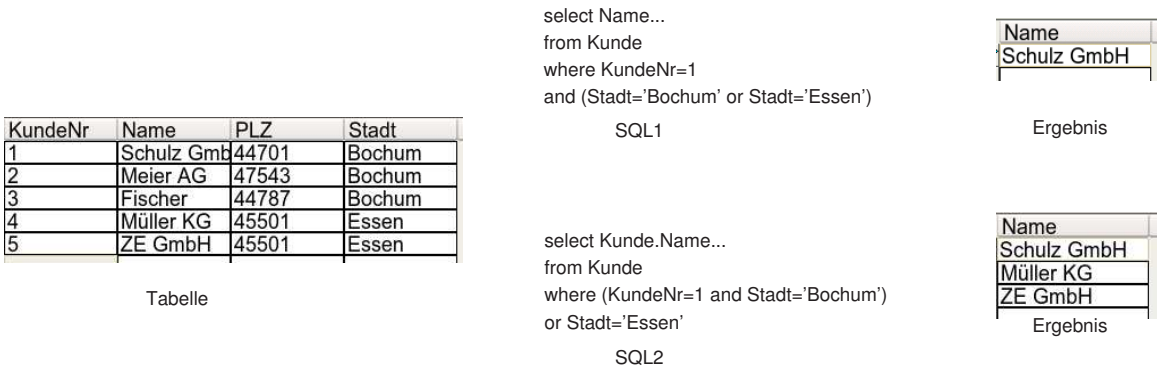


Abbildung 5.2: Ausgabe von SQL 5.10 und SQL 5.11

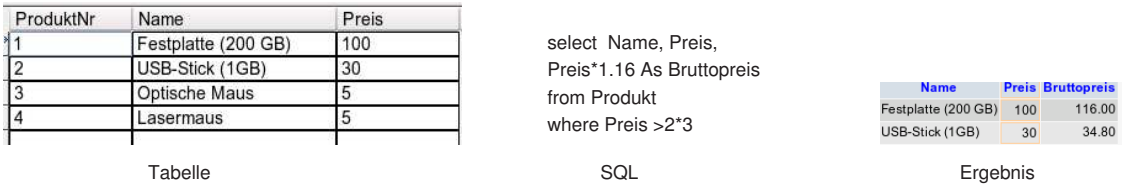


Abbildung 5.3: Ausgabe von SQL 5.12

```
select  Name, Preis,
Preis*1.16 As Bruttopreis
from Produkt
where Preis >2*3
```

SQL unterstützt Funktionen. SQL 5.13 und Abb. 5.4 zeigen dies. Darüber hinaus sehen Sie, dass wir Apostrophe benutzen müssen, wenn unsere Spaltenüberschriften Leerzeichen enthalten sollen.

SQL 5.13 *SQL mit Funktionen*

```
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 Produkt
```

Eine weitere wichtige Eigenschaft von SQL ist die Unterstützung von Wildcards. Wildcards sind Platzhalterzeichen, die den Wert beliebiger Zeichen annehmen können. Na ja, ist vielleicht nicht wirklich prickelnd ausgedrückt, aber anhand eines Beispiels wird die Bedeutung des Begriffes sofort klar. % ist in SQL das Zeichen für beliebig viele beliebige Zeichen[4]. B% bedeutet: Fängt mit B an, dann kommen beliebig viele andere Zeichen, B% trifft also Bernd, Blümel, Berning, Boris, Becker usw.. B%n% bedeutet: Fängt mit B an, dann kommt irgendwann ein n, dann wieder beliebige andere Zeichen, trifft also von der obigen Auswahl nur noch Bernd und Berning. B%n%d bedeutet: Fängt mit B an, dann kommt irgendwann ein n, dann wieder beliebige andere Zeichen,dann ein d und danach nichts mehr, trifft also von der obigen Auswahl nur noch Bernd.

[4]Beachten Sie, dass das in Access nicht gilt, Access ist hier nicht standardkonform, in Access ist * dieses Zeichen, warum auch immer.

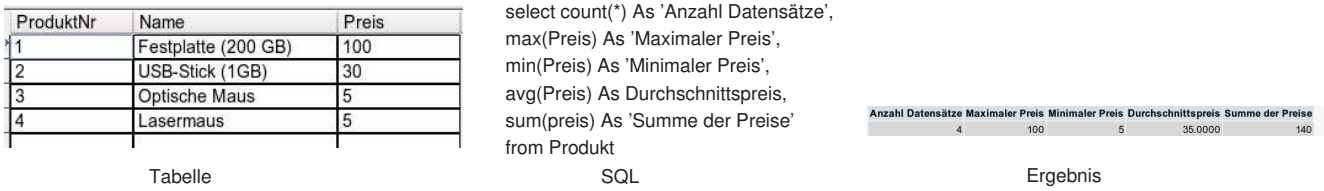


Abbildung 5.4: Ausgabe von SQL 5.13

SQL 5.14 sucht also alle Datensätzen der Tabelle Kunde, in deren Namen der Buchstabe i vorkommt. Abb. 5.5 zeigt das Resultat. Beachten Sie, dass Sie bei Wildcard-Vergleichen nie das Gleichheitszeichen (=) verwenden dürfen, sondern immer das SQL-Schlüsselwort „like“. Die Zeichenkette, mit der verglichen wird, muss in Apostrophe gesetzt werden. Wildcard-Vergleiche sind nur in Zeichenketten-Feldern[5] erlaubt.

SQL 5.14 *SQL mit Wildcard*

```
select Name
from Kunde
where name like '%i%'
```

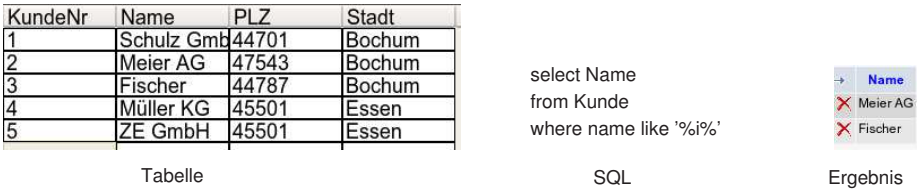


Abbildung 5.5: Ausgabe von SQL 5.14

5.2 Gruppierung

In SQL können Datensätze aggregiert werden. Ich zeige dies sofort an einem Beispiel.

SQL 5.15 *SQL mit Gruppierung*

```
select Stadt, count(*) as Anzahl
from Kunde
Group by Stadt
```

Das Ergebnis ist in Abb. 5.6 dargestellt.

Wir haben in der ursprünglichen Tabelle drei Datensätze mit Kunden aus Bochum (Wert des Attributs Stadt ist Bochum), zwei Datensätze mit Kunden aus Essen (Wert des Attributs Stadt ist Essen). Wird nach einem Attribut gruppiert (SQL-Anweisung „Group By“), so ermittelt das Datenbanksystem die unterschiedlichen Ausprägungen des Attributs nach dem gruppiert wird. In unserem Beispiel wird nach Stadt gruppiert. Die unterschiedlichen Ausprägungen sind Bochum

[5] Also nicht bei Zahlen.

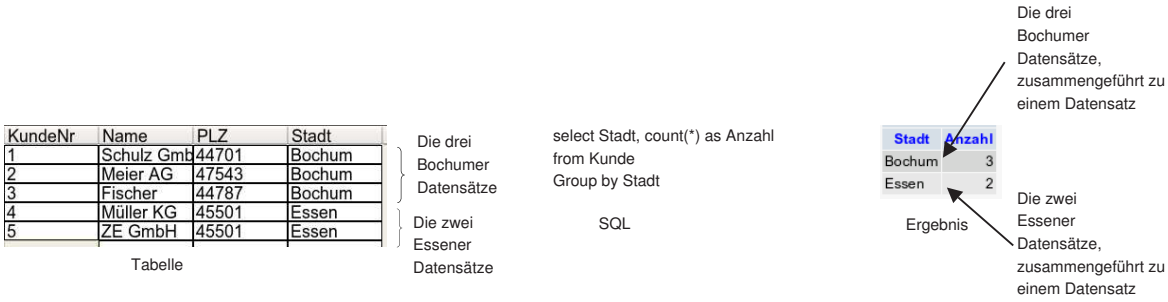


Abbildung 5.6: Ausgabe von SQL 5.15

und Essen. Dann werden die Datensätze, wo die Ausprägungen des Gruppierungsattributs übereinstimmen, zu einem Datensatz zusammengeführt[6]. In unserem Beispiel werden die Datensätze 1, 2 und 3 (die Ausprägung des Gruppierungsattributs Stadt ist bei allen diesen Datensätzen gleich, nämlich Bochum) sowie die Datensätze 4 und 5 (die Ausprägung des Gruppierungsattributs Stadt ist bei allen diesen Datensätzen gleich, nämlich Essen) 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. Betrachten wir SQL 5.16 und das in Abb. 5.7 dargestellte Ergebnis.

SQL 5.16 *SQL mit Gruppierung (falsche Attribute)*

```
select Name, Stadt, count(*) as Anzahl
from Kunde
Group by Stadt
```

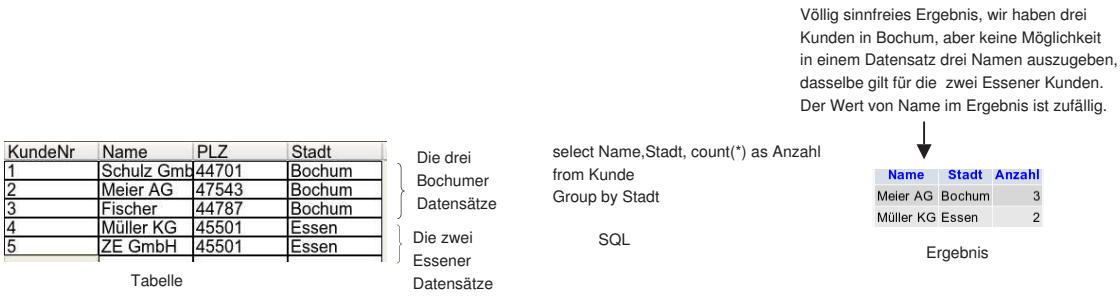


Abbildung 5.7: Ausgabe von SQL 5.16

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 Stadt. Wie kann das Datenbanksystem aber mehrere unterschiedliche Namen in einem Datensatz darstellen? Die Antwort ist: Gar nicht. Das Datenbanksystem würfelt und stellt einen zufälligen 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 Stadt).
- Die in SQL 5.13 gezeigten Funktionen. So würde z.B. max(KundeNr) im Select-Teil Sinn machen, denn das Datenbanksystem kann in den Ergebnisdatensätzen die größte KundeNr darstellen (in unserem Beispiel 3 für Bochum und 5 für Essen).

[6] Gruppierung nach dem Primärschlüssel macht daher wenig Sinn. Wissen auch Sie, warum :-))?

Zusätzlich zu denen in Kapitel 5.1 dargestellten Möglichkeiten, Bedingungen in Abfragen einzubauen, kann man auch Bedingungen an die Gruppierung knüpfen. Betrachten wir dazu SQL 5.17 und 5.18, sowie das in Abb. 5.7 dargestellte Ergebnis.

SQL 5.17 *SQL mit Gruppierung und Einschränkung 1*

```
select Stadt, count(*) as Anzahl, max(KundeNr) As 'Größte Kundennummer'
from Kunde
Group by Stadt
having count(*) > 2
```

SQL 5.18 *SQL mit Gruppierung und Einschränkung 2*

```
select Stadt, count(*) as Anzahl,  max(KundeNr) As 'Größte Kundennummer'
from Kunde
Group by Stadt
having max(KundeNr) > 3
```

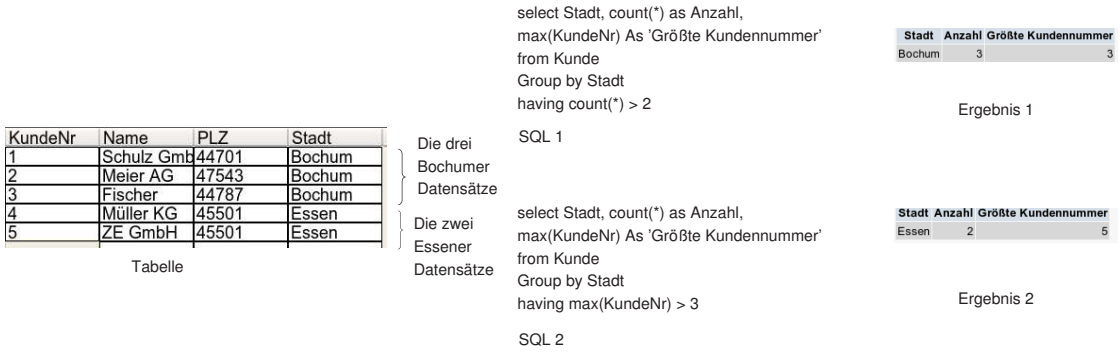


Abbildung 5.8: Ausgabe von SQL 5.17 und 5.18

Einschränkungen erfolgen also mit dem SQL-Schlüsselwort „having“. Danach können alle in SQL 5.13 betrachteten Funktionen genutzt werden.

5.3 Abfragen über mehrere Tabelle

Da wir ja 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.

Glücklicherweise ist dies aber auch nicht zu kompliziert. Am Grundgerüst der Select-Abfragen ändert sich nichts. Wir beginnen auch hier mit einem Beispiel. Wir möchten uns Lieferdatum und das Datum der Auftragserteilung des Kunden Meier AG anzeigen lassen. Der Name des Kunden, PLZ und Stadt sollen ebenfalls mit ausgegeben werden.

SQL 5.19 *SQL über mehrere Tabellen (Kunde-Auftrag 1)*

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

Wir sehen:

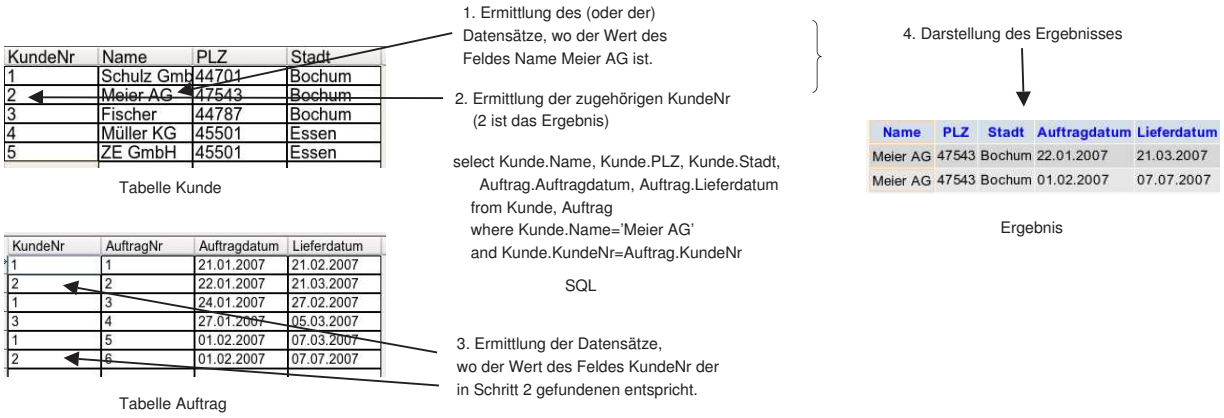
- Im „Select-Teil“ schreiben wir den Tabellennamen vor die Namen der Felder. Dies ist aber ziemlich einleuchtend. Wir haben ja jetzt Abfragen auf mehrere Tabellen. Innerhalb einer Tabelle müssen die Namen der Felder natürlich eindeutig sein (Wir können nicht zwei Felder gleichen Namens in einer Tabelle haben). Bei mehreren Tabellen ist das nicht mehr so. In unserem Beispiel gibt es ein Attribut Name z.B. sowohl in der Kunde-, als auch in der Produkt-Tabelle. Der Tabellenname zusammen mit dem Feldnamen hingegen ist wieder eindeutig. Der Feldname wird, wie SQL 5.19 entnommen werden kann, 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. Dies z.B. in SQL 5.20 der Fall. Wir lassen die Kundeninformationen dort nicht mit ausgeben. Dennoch muss die Tabelle Kunde in den „From-Teil“ mit aufgenommen werden, da diese Tabelle zur Ermittlung des Ergebnisses notwendig ist.
- 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 KundenNr (in unserem Beispiel 2).
  3. Gehe mit dieser KundeNr in die Tabelle Auftrag und ermittle die Datensätze in Auftrag, wo die KundeNr in Auftrag (der Fremdschlüssel) mit der ermittelten KundeNr in Kunde übereinstimmt.
  4. Gib die zugehörigen Datensätze aus.

Abb. 5.9 zeigt dies zusammenfassend.

SQL 5.20 *SQL über mehrere Tabellen (Kunde-Auftrag 2)*

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



KundeNr	Name	PLZ	Stadt
1	Schulz Gmb	44701	Bochum
2	Meier AG	47543	Bochum
3	Fischer	44787	Bochum
4	Müller KG	45501	Essen
5	ZE GmbH	45501	Essen

KundeNr	AuftragNr	Auftragdatum	Lieferdatum
1	1	21.01.2007	21.02.2007
2	2	22.01.2007	21.03.2007
1	3	24.01.2007	27.02.2007
3	4	27.01.2007	05.03.2007
1	5	01.02.2007	07.03.2007
2	6	01.02.2007	07.07.2007

Name	PLZ	Stadt	Auftragdatum	Lieferdatum
Meier AG	47543	Bochum	22.01.2007	21.03.2007
Meier AG	47543	Bochum	01.02.2007	07.07.2007

Abbildung 5.9: Ausgabe und Vorgehensweise von SQL 5.19

Gehen wir zum nächsten Beispiel über: Wir möchten Name, PLZ und Stadt aller Kunden ausgeben, die am 07.03 beliefert werden sollen.

SQL 5.21 *SQL über mehrere Tabellen (Kunde-Auftrag 3)*

```
select Kunde.Name, Kunde.PLZ, Kunde.Stadt
from Kunde, Auftrag
where Auftrag.Lieferdatum='07.03.2007'
and Kunde.KundeNr=Auftrag.KundeNr
```

Hier erkläre ich nur noch den „Bedingungsteil“. In Worten heißt die im „Bedingungsteil“ formulierte Anweisung an das Datenbanksystem:

1. Ermittle die Datensätz in der Tabelle Auftrag, wo das Feld Lieferdatum den Wert „07.03.2007“ hat.
2. Ermittle in diesen Datensätzen die KundenNr's (in unserem Beispiel 1und 2).
3. Gehe mit dieser KundeNr in die Tabelle Kunde und ermittle die Datensätze in Kunde, wo die ermittelte KundeNr in Auftrag (der Fremdschlüssel) mit der KundeNr (dem Primärschlüssel) in Kunde übereinstimmt.
4. Gib die ermittelten Informationen aus.

Abb. 5.10 zeigt dies zusammenfassend.

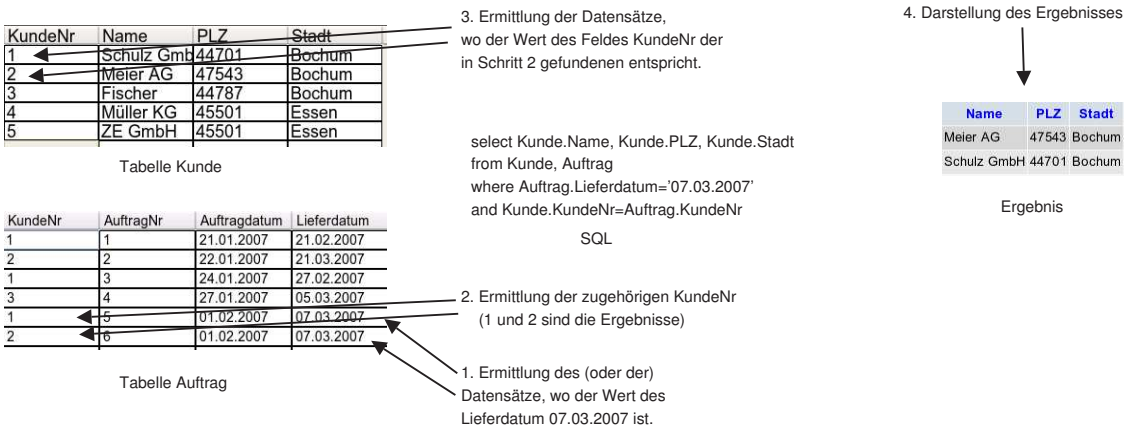


Abbildung 5.10: Ausgabe und Vorgehensweise von SQL 5.21

SQL 5.19 bis SQL 5.21 können seit dem SQL-2-Standard alternativ formuliert werden. Ich zeige dies an SQL 5.21.

SQL 5.22 *SQL über mehrere Tabellen, alternative Realisierung der Primärschlüssel - Fremdschlüsselbeziehung*

```
select Kunde.Name, Kunde.PLZ, Kunde.Stadt
from Auftrag
join Kunde on (Kunde.KundeNr=Auftrag.KundeNr)
where Auftrag.Lieferdatum='07.03.2007'
```

Sie sehen, dass die Primärschlüssel - Fremdschlüsselbeziehung nicht mehr durch „and“ an die Bedingung angeschlossen wird, sondern mit dem Konstrukt „join on“. Welche Tabelle in den „From-Teil“ bzw. in das „join on“ aufgenommen wird, spielt keine Rolle. SQL 5.22 ist äquivalent zu SQL 5.23[7].

SQL 5.23 *SQL über mehrere Tabellen, alternative Realisierung der Primärschlüssel - Fremdschlüsselbeziehung (2)*

```
select Kunde.Name, Kunde.PLZ, Kunde.Stadt
from Kunde
join Auftrag on (Kunde.KundeNr=Auftrag.KundeNr)
where Auftrag.Lieferdatum='07.03.2007'
```

[7] Und selbstverständlich zu SQL 5.21.

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 ja 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“ verbunden die Primärschlüssel - Fremdschlüsselbeziehungen aufgenommen. Ich zeige 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 verbunden.

Die Lösung lautet:

SQL 5.24 *SQL über mehrere Tabellen, mehr als zwei Tabellen beteiligt*

```
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
```

Am „Select-“ und “From-Teil“ ändert sich von der Logik her nichts. Daher erkläre ich nur noch den „Bedingungsteil“. Da die darzustellende Information jetzt auf drei Tabellen verteilt ist, benötigen wir im Bedingungsteil nun zwei Zeilen zur Abbildung der Primärschlüssel - Fremdschlüsselbeziehungen. In Worten heißt die im „Bedingungsteil“ formulierte Anweisung an das Datenbanksystem:

1. Ermittle den (oder die) Datensätze der Tabelle Produkt, wo 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.

Abb. 5.11 zeigt dies zusammenfassend.

Abschliessend wollen wir noch die Namen der Kunden, in deren Aufträgen das Produkt 'USB-Stick (1GB)' enthalten ist, ausgeben. Dies ist eine einfache Erweiterung von SQL 5.24.

SQL 5.25 *SQL über mehrere Tabellen, mehr als zwei Tabellen beteiligt (2)*

```
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
```

Die Erläuterung und das Ergebnis ist in Abb. 5.12 dargestellt.

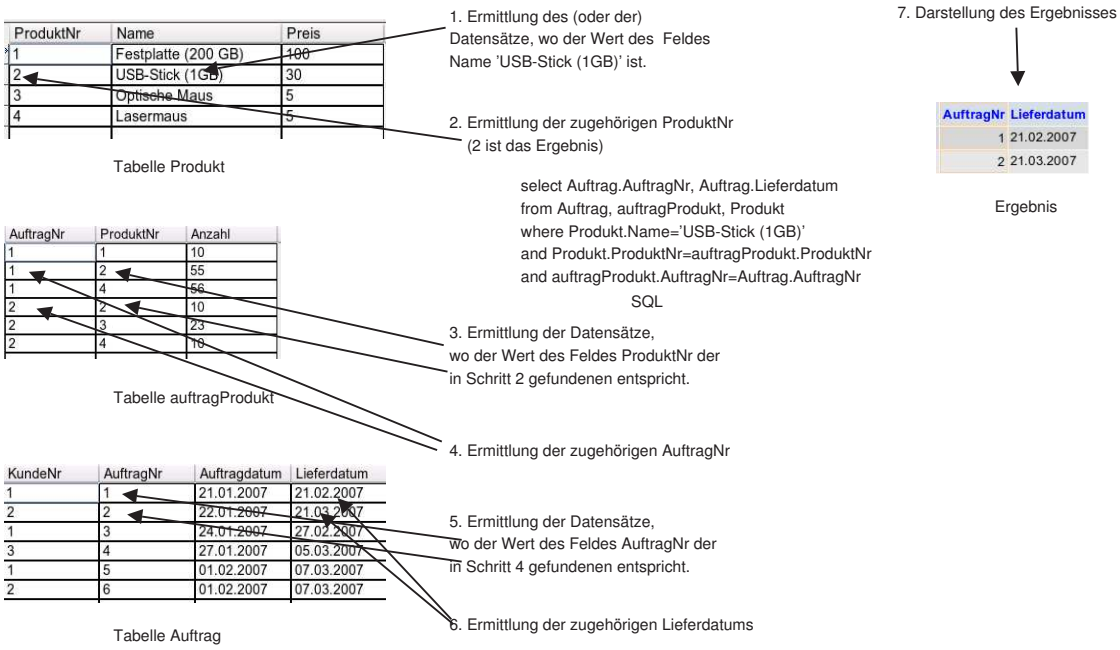


Abbildung 5.11: Ausgabe und Vorgehensweise von SQL 5.24

5.4 Left Join, Right Join, Null

Starten wir auch hier mit einem Beispiel: Wir möchten Name, PLZ und Stadt aller Kunden ausgeben, die gerade einen Auftrag offen haben. Das können Sie, wenn Sie Kapitel 5.3 verstanden haben, relativ leicht lösen. Wir betrachten SQL 5.20 bzw. 5.21. Hier müssen wir ja nur die Einschränkung, das Lieferdatum betreffend, weglassen und wir haben die Lösung (vgl. SQL 5.26).

SQL 5.26 *Alle Kunden, die Aufträge offen haben*

```
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)
```

Abb. 5.13 zeigt die grafische Darstellung.

Nun erweitern wir die Aufgabenstellung: Wir wollen alle Kundendatensätze in der Ausgabe erscheinen lassen, egal, ob sie einen Auftrag erteilt 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 KundenNr 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 demzufolgen die AuftragNr nicht anzeigen.

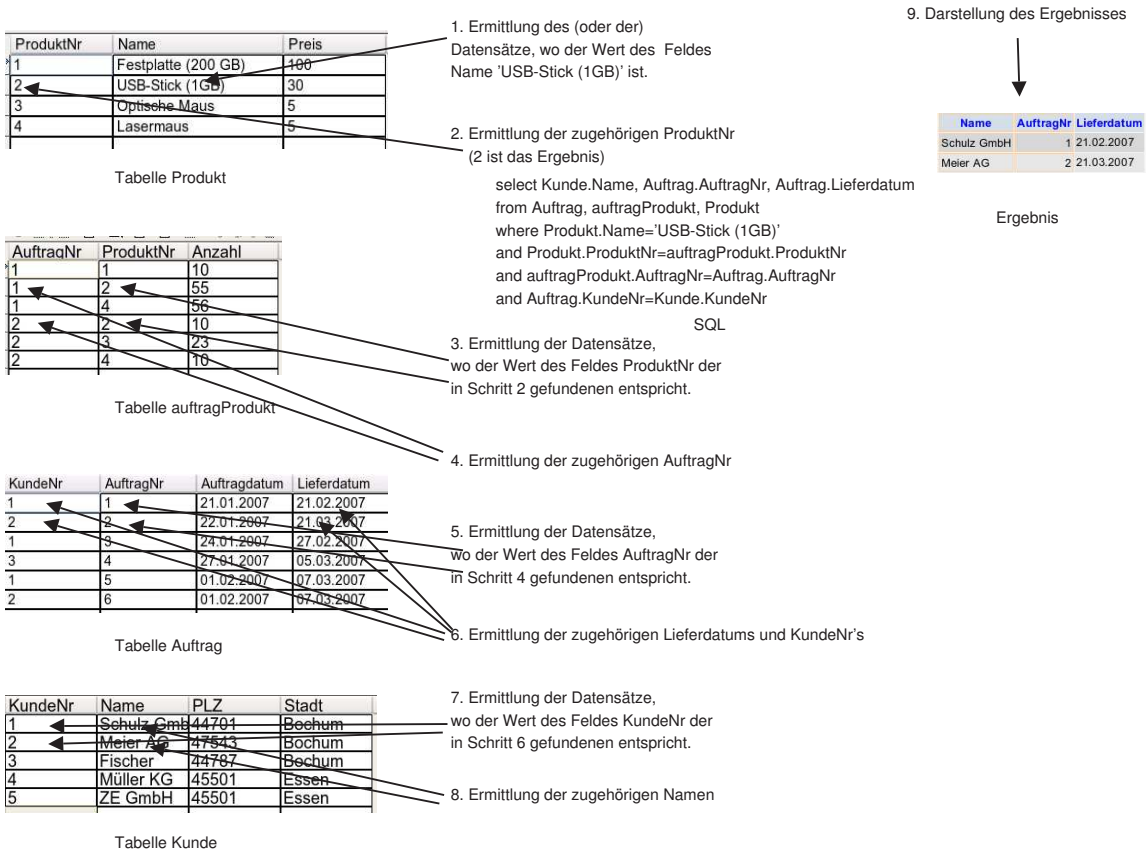


Abbildung 5.12: Ausgabe und Vorgehensweise von SQL 5.25

Lösungen 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 andersherum. SQL 5.27 zeigt die Lösung.

SQL 5.27 *Left Join, Right Join*

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

oder alternativ

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

Left Joins werden also genau wie der normale Join in SQL 5.21 aufgebaut. Vor das SQL-Schlüsselwort Join wird einfach nur Left geschrieben, alles andere ist identisch. Das Resultat ist in Abb. 5.14 dargestellt.

Bei Kunden, zu denen keine Aufträge existieren, wird als AuftragNr „Null“ eingetragen (vgl. Abb. 5.14). „Null“ ist in der Datenbanktheorie ein bestimmter Wert. „Null“ bedeutet kein Wert, nicht einmal der leere String[8].

[8] die leere Zeichenkette

KundeNr	Name	PLZ	Stadt
1	Schulz Gmb	44701	Bochum
2	Meier AG	47543	Bochum
3	Fischer	44787	Bochum
4	Müller KG	45501	Essen
5	ZE GmbH	45501	Essen

Tabelle Kunde

KundeNr	AuftragNr	Auftragdatum	Lieferdatum
1	1	21.01.2007	21.02.2007
2	2	22.01.2007	21.03.2007
1	3	24.01.2007	27.02.2007
3	4	27.01.2007	05.03.2007
1	5	01.02.2007	07.03.2007
2	6	01.02.2007	07.03.2007

Tabelle Auftrag

```
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)
```

SQL

Name	PLZ	Stadt	AuftragNr
Meier AG	47543	Bochum	2
Meier AG	47543	Bochum	6
Schulz GmbH	44701	Bochum	1
Schulz GmbH	44701	Bochum	3
Schulz GmbH	44701	Bochum	5
Fischer	44787	Bochum	4

Ergebnis

Abbildung 5.13: Ausgabe von SQL 5.26

KundeNr	Name	PLZ	Stadt
1	Schulz Gmb	44701	Bochum
2	Meier AG	47543	Bochum
3	Fischer	44787	Bochum
4	Müller KG	45501	Essen
5	ZE GmbH	45501	Essen

Tabelle Kunde

KundeNr	AuftragNr	Auftragdatum	Lieferdatum
1	1	21.01.2007	21.02.2007
2	2	22.01.2007	21.03.2007
1	3	24.01.2007	27.02.2007
3	4	27.01.2007	05.03.2007
1	5	01.02.2007	07.03.2007
2	6	01.02.2007	07.03.2007

Tabelle Auftrag

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

oder alternativ

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

SQL

Name	PLZ	Stadt	AuftragNr	
Meier AG	47543	Bochum	2	Kunden mit Aufträgen
Meier AG	47543	Bochum	6	
Schulz GmbH	44701	Bochum	1	
Schulz GmbH	44701	Bochum	3	
Schulz GmbH	44701	Bochum	5	
Fischer	44787	Bochum	4	
Müller KG	45501	Essen	NULL	Kunden ohne Aufträgen
ZE GmbH	45501	Essen	NULL	

Ergebnis

Abbildung 5.14: Ausgabe von SQL 5.27

Mit diesem Konstrukt können wir nun auch Fragen in die andere Richtung beantworten. Damit meine ich:

- Es ist einfach SQL-Anweisungen zu schreiben, für Fragen wie: Zu welchen Kunden existieren Aufträge? Dies ist ja SQL 5.26.
- Aber was ist mit: Zu welchen Kunden existieren keine Aufträge?

Die SQL-Anweisung zu dieser Fragestellung ist nämlich **nicht** SQL 5.28[9].

SQL 5.28 *SQL-Versuch Kunden ohne Aufträge - leider falsch*

```
select Kunde.Name, Kunde.PLZ, Kunde.Stadt,
       Auftrag.AuftragNr
from Kunde, Auftrag
where not (Kunde.KundeNr=Auftrag.KundeNr)
```

Das Ergebnis der Abfrage SQL 5.28 ist in Abb. 5.15 dargestellt.

Sie sehen, SQL 5.28 gibt aus, welcher Kunde welchen Auftrag nicht erteilt hat. Zunächst werden die vier unserer fünf Kunden angezeigt, die Auftrag 1 nicht erteilt haben, dann kommen die vier Kunden, die Auftrag 2 nicht erteilt haben, usw.. Und wenn wir mal ganz ehrlich zu uns sind, das ist auch genau der Inhalt von SQL 5.28[10].

Aber da wir jetzt Left bzw. Right Joins beherrschen, können wir auch diese Aufgabe lösen. Wir formulieren ein SQL der Art:

[9] Die AuftragNr ist in die Ausgabe aufgenommen, damit wir das Ergebnis der SQL-Abfrage gleich richtig interpretieren können.

[10] Ja von Logik kann man einen Knoten im Gehirn bekommen.

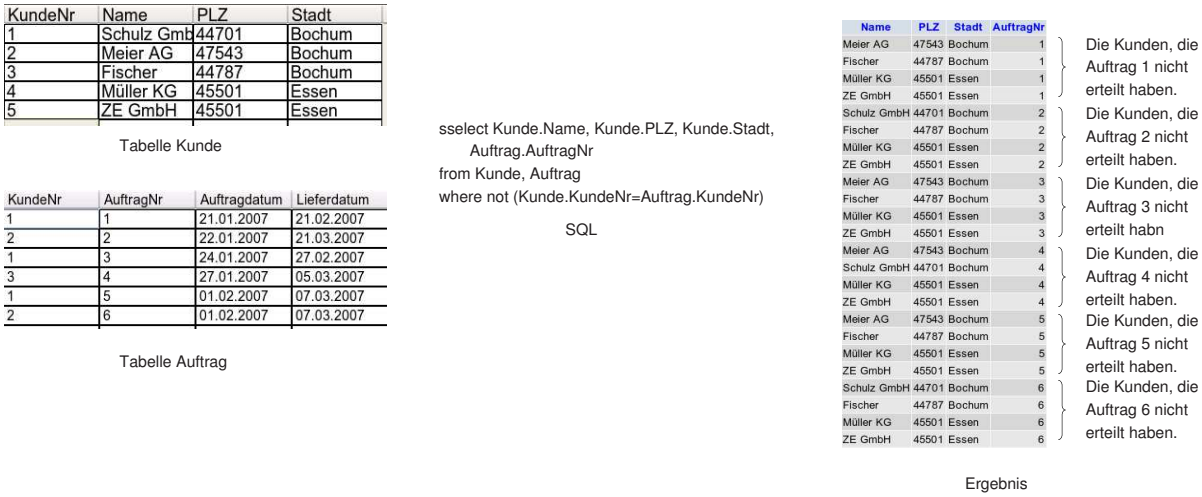


Abbildung 5.15: Ausgabe von SQL 5.28

1. Zunächst führen wir einen Left Join von Kunde auf Auftrag durch (wie in SQL 5.27). Kunden, die keinen Auftrag erteilt haben, erhalten, wie wir bereits wissen, als AuftragNr „Null“ eingetragen.
2. Dann sagen wir dem Datenbanksystem, es soll die Datensätze der Ergebnismenge suchen, die als AuftragNr „Null“ eingetragen haben, und diese ausgeben.
3. Wir lassen AuftragNr aus der Ausgabe weg.

SQL 5.29 zeigt die Lösung.

SQL 5.29 *SQL-Abfrage Kunden ohne Aufträge*

```
select Kunde.Name, Kunde.PLZ, Kunde.Stadt
from Kunde
left join Auftrag on (Kunde.KundeNr=Auftrag.KundeNr)
where Auftrag.AuftragNr is null
```

Das einzig Gewöhnungsbedürftige hier ist, das wir auf Null nicht, wie man es annehmen sollte mit einer Zeile, wie

```
where Auftrag.AuftragNr = null
```

abfragen, sondern eben mit

```
where Auftrag.AuftragNr is null.
```

Aber das muss man sich einfach merken. Das Resultat zeigt Abb. 5.16. Es ist genau das, was wir wollen.

5.5 Delete, Update und Insert

Bislang haben wir die SQL-Anweisungen, um Informationen der Datenbank anzuzeigen, besprochen. Selbstverständlich gibt es auch SQL-Kommandos, die Datensätze löschen, verändern oder einzufügen. Diese Kommandos wirken immer nur auf eine Tabelle. Wir beginnen mit dem Löschen.

SQL 5.30 *SQL-Abfrage Kunden löschen*

```
delete
from Kunde
```

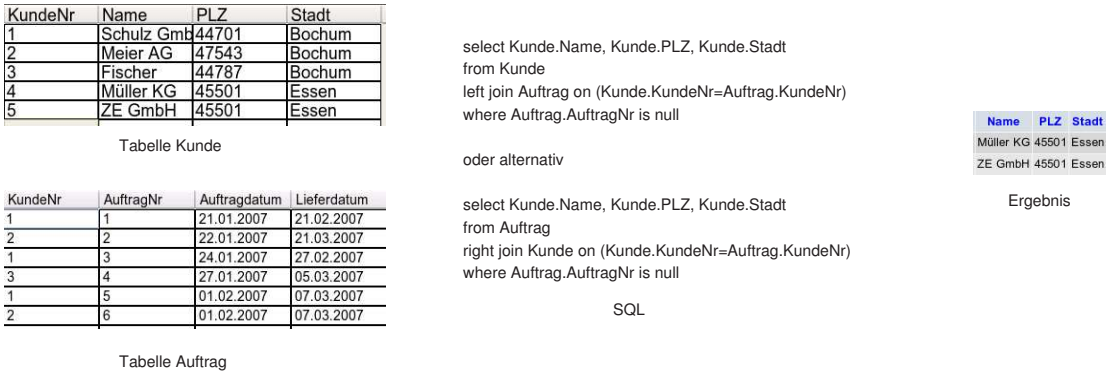


Abbildung 5.16: Ausgabe von SQL 5.29

SQL 5.30 löscht alle Datensätze der Kundentabelle. Das SQL-Kommando „delete“ funktioniert genauso, wie selects auf eine Tabelle, mit dem Unterschied:

- Datensätze werden gelöscht, nicht angezeigt :-).
- Hinter „delete“ dürfen keine Feldnamen stehen, es wird immer der ganze Datensatz gelöscht.

SQL 5.31 *SQL-Abfrage Essener Kunden löschen*

```
delete
from Kunde
where Stadt='Essen'
```

SQL 5.31 löscht alle Essener Kunden.

SQL 5.32 *SQL-Delete mit Wildcard*

```
delete
from Kunde
where Stadt='Essen'
and Name like 'M%'
```

SQL 5.32 löscht alle Essener Kunden, deren Name mit M beginnen. Ich verzichte hier auf weitergehende Diskussionen, im Bedingungsteil ist alles erlaubt, was in Kapitel 5.1 vorgestellt wurde.

Kommen wir nun zum Update.

SQL 5.33 *Kunden ändern*

```
update Kunde
set Stadt='Essen'
```

Alle Kunden wohnen nun in Essen.

SQL 5.34 *Kunden ändern (2)*

```
update Kunde
set Stadt='Essen',
    Name='Blümel'
```

Alle Kunden wohnen in Essen und heißen Blümel[11].

[11] Wie langweilig ...

SQL 5.35 *Kunden ändern (3)*

```
update Kunde
set Stadt='Essen',
    Name='Blümel'
where Stadt='Bochum'
and Name like 'M%'
```

Alle Bochumer Kunden, deren Name mit M anfängt, ziehen nach Essen und heißen fortan Blümel. Auch hier ist im Bedingungsteil alles erlaubt, was in Kapitel 5.1 vorgestellt wurde, so dass sich auch hier eine weitere Diskussion dieser Thematik erübrigt. Updates können auch rechnen:

SQL 5.36 *Preise erhöhen*

```
update Produkt
set Preis=1.1*Preis
```

Abschließend behandeln wir das Insert Kommando. Auch hier tun wir das anhand eines Beispiels:

SQL 5.37 *Neues Produkt eintragen*

```
insert into Produkt
    (ProduktNr, Name, Preis)
values
    (5, 'Funktastatur', 45)
```

Wenn der Primärschlüssel einer zu einer Entity-gehörenden Tabelle für jeden Datensatz um Eins hochgezählt wird, kann man diesem in vielen Datenbanksystemen die Eigenschaft „auto_increment“ zuweisen[12]. Dann muss der Wert des Primärschlüssels nicht angegeben werden, das Datenbanksystem vergibt selbstständig die nächste freie Zahl (vgl. SQL 5.38).

SQL 5.38 *Neues Produkt eintragen, Primärschlüssel mit auto_increment-Eigenschaft*

```
insert into Produkt
    (ProduktNr, Name, Preis)
values
    ('', 'Funktastatur', 45)
```

[12] Autowert in Access.