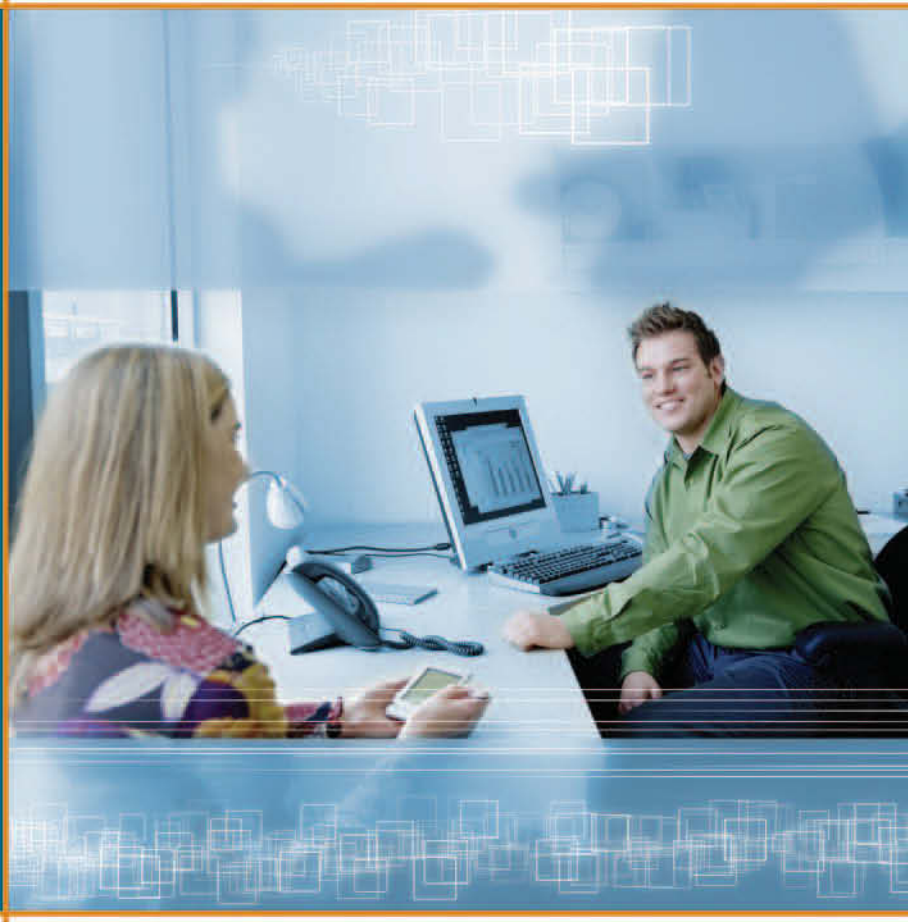
MySQL Press



LUKE WELLING
LAURA THOMSON

MySQL® Tutorial

Die kompakte Einführung in die Arbeit mit MySQL

ADDISON-WESLEY

Teil II

Datenbanken mit MySQL entwerfen und erstellen

3

Crashkurs zum Datenbankentwurf

In diesem Kapitel werden wir die grundlegenden Prinzipien des Datenbankentwurfs und der Normalisierung vorstellen. Eine gut entworfene Datenbank minimiert Redundanzen ohne Datenverlust. Das bedeutet, dass unser Ziel darin besteht, den geringst möglichen Speicherplatz für unsere Datenbank zu verbrauchen, während wir gleichzeitig für die volle Verknüpfung der Daten sorgen.

Das Kapitel umfasst die folgenden Themen:

- Datenbankkonzepte und -terminologie
- Prinzipien des Datenbankentwurfs
- Normalisierung und Normalformen
- Übungen zum Datenbankentwurf

3.1 Datenbankkonzepte und -terminologie

Um die in diesem Kapitel behandelten Prinzipien verstehen zu können, müssen Sie sich zuerst einige Grundkonzepte und Terminologien aneignen.

3.1.1 Entitäten und Relationen

Die Grundlage aller Datenbankmodelle sind die Entitäten und Relationen. Entitäten sind Einheiten in der praktischen Welt, über die wir Informationen in der Datenbank speichern möchten. Nehmen wir als Beispiel einmal an, dass wir Informationen zu Angestellten und den Abteilungen, in denen diese tätig sind, speichern möchten. In diesem Fall stellt ein Angestellter eine Entität und eine Abteilung eine weitere Entität dar. Diese Entitäten können untereinander verknüpft werden. Beispielsweise arbeitet ein Angestellter für eine Abteilung. Die Verknüpfung zwischen den Entitäten des Angestellten und der Abteilung lautet dann »arbeitet-für«.

Verknüpfungen können in verschiedenen Ausprägungen auftreten. Sie können in der Form von Eins-zu-Eins, Eins-zu-Viele (oder Viele-zu-Eins in Abhängigkeit von Ihrer Blickrichtung) oder Viele-zu-Viele erscheinen. Eine Eins-zu-Eins-Verknüpfung verknüpft genau zwei Entitäten miteinander. Wenn jeder einzelne Angestellte in der Organisation über einen Ar-

beitsplatz mit Schreibtisch verfügt, dann würde dies eine Eins-zu-Eins-Verknüpfung darstellen. Die Verknüpfung »arbeitet-für« bezeichnet in unserem Beispiel üblicherweise eine Viele-zu-Eins-Verknüpfung. Dies bedeutet, dass viele Angestellte für eine einzelne Abteilung arbeiten, während jeder einzelne Angestellte aber für nur eine Abteilung tätig ist. Diese beiden Verknüpfungen sind in der Abbildung 3.1 dargestellt.

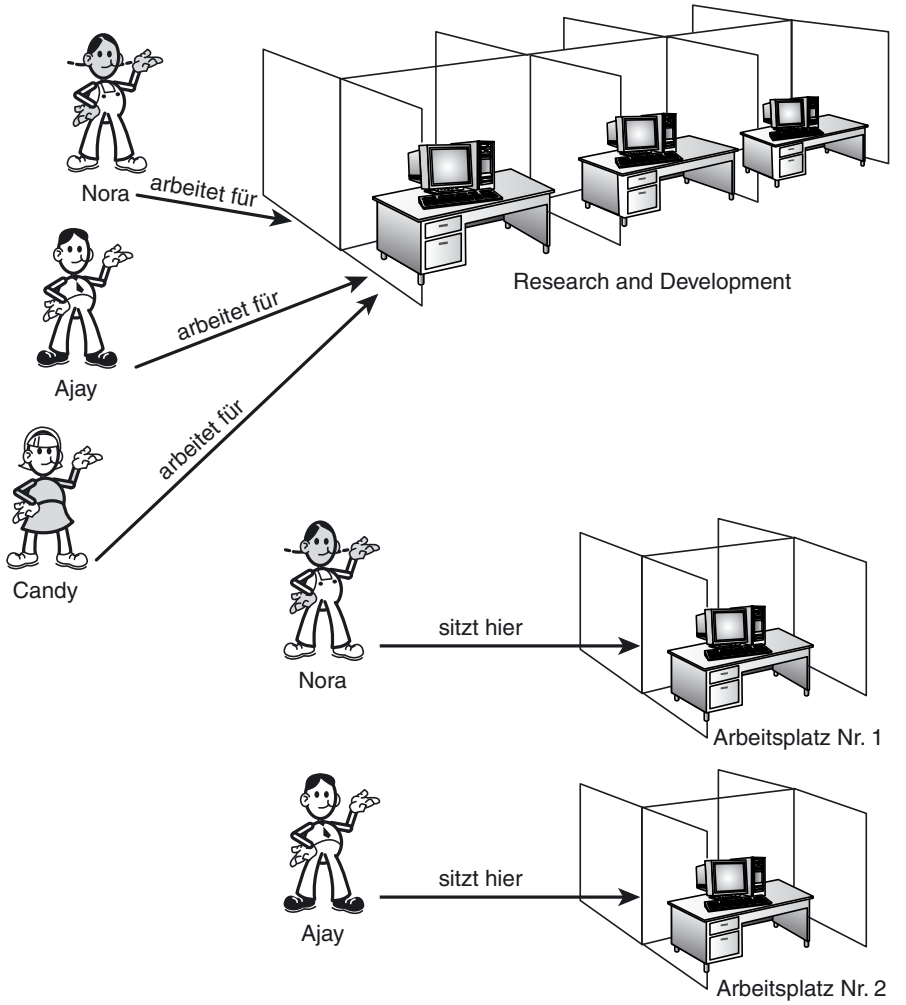


Abbildung 3.1 Die Verknüpfung »sitzt hier« ist eine Eins-zu-Eins-Verknüpfung. Die Verknüpfung »arbeitet für« ist eine Viele-zu-Eins-Verknüpfung.

Beachten Sie, dass die Entitäten, Verknüpfungen und Ausprägungen der Verknüpfungen von Ihrer Arbeitsumgebung und dem Modell abhängen, das Sie abbilden möchten. In manchen Unternehmen beispielsweise arbeiten die Angestellten für mehrere Abteilungen. In diesem Fall würde die Verknüpfung »arbeitet-für« zu einer Viele-zu-Viele-Verknüpfung werden. Wenn

sich jemand einen Arbeitsplatz mit Schreibtisch mit anderen Personen teilt, oder wenn jemand anstelle eines Arbeitsplatzes über ein eigenes Büro verfügt, wäre die Verknüpfung »sitzt-hier« keine Eins-zu-Eins-Verknüpfung mehr.

Beim Entwurf eines Datenbankmodells müssen Sie daher die für Ihre Umgebung geltenden Regeln beim Entwurf Ihres Modells berücksichtigen. Keine zwei Modelle sind exakt gleich.

3.1.2 Relationen oder Tabellen

MySQL ist ein relationales Datenbankmanagementsystem (RDBMS). Das bedeutet, es unterstützt Datenbanken, die aus einer Reihe von Relationen bestehen. Eine Relation stellt in diesem Sinne eine Tabelle mit Daten dar. Beachten Sie an dieser Stelle, dass die Begriffe *Tabelle* und *Relation* ein und dasselbe meinen. In diesem Buch werden wir den gebräuchlicheren Begriff Tabelle verwenden. Falls Sie schon mal mit einer Tabellenkalkulation gearbeitet haben, wissen Sie, dass jedes Arbeitsblatt der Tabellenkalkulation typischerweise eine Tabelle mit Daten darstellt. Die Abbildung 3.2 enthält eine Beispieltabelle.

employee

employeeID	name	job	departmentID
7513	Nora Edwards	Programmer	128
9842	Ben Smith	DBA	42
6651	Ajay Patel	Programmer	128
9006	Candy Burnett	Systems Administrator	128

Abbildung 3.2 Die Tabelle *employee* speichert die Angestellten-IDs (*employeeID*), die Namen (*name*), die Jobs (*job*) und die Abteilung (*departmentID*), für die jeder Angestellte tätig ist.

Wie Sie sehen, speichert diese Tabelle Daten über die Angestellten eines bestimmten Unternehmens (wir haben hier nicht die gesamten Angestelltendaten, sondern nur einen Auszug daraus aufgeführt).

3.1.3 Spalten und Attribute

Jede *Spalte* bzw. jedes *Attribut* in Datenbanktabellen beschreibt die Daten, aus denen jeder einzelne Datensatz besteht. Die Begriffe Spalte und Attribut werden oft gleichbedeutend verwendet. Tatsächlich ist aber eine Spalte Bestandteil einer Tabelle, während Attribute sich auf Entitäten aus der praktischen Welt beziehen, die durch die Tabelle abgebildet werden. In Abbildung 3.2 können Sie erkennen, dass jeder Angestellte über eine employeeID, name, job und departmentID verfügt. Diese bilden die Spalten der Tabelle *employee*, die man auch als die Attribute der Tabelle *employee* bezeichnen kann.

3.1.4 Zeilen, Datensätze und Tupel

Schauen Sie sich nun noch einmal die Tabelle *employee* an. Jede Zeile in der Tabelle repräsentiert einen einzelnen Angestellten-Datensatz. Vielleicht haben Sie schon davon gehört, dass man hierzu Zeile, Datensatz oder auch Tupel sagt. Jede Zeile in der Tabelle besteht aus Spalten mit Werten für jede einzelne Spalte.

3.1.5 Schlüssel

Ein Superkey ist eine Spalte (bzw. ein Satz von Spalten), die zur eindeutigen Identifizierung einer Zeile in der Tabelle verwendet werden kann. Ein Schlüssel ist ein minimaler Superkey. Schauen Sie sich beispielsweise einmal die Tabelle *employee* an. Mit *employeeID* und *name* zusammen können Sie jede beliebige Zeile in der Tabelle identifizieren. Sie könnten ebenso auch einen Satz aus sämtlichen Spalten (*employeeID*, *name*, *job*, *departmentID*) verwenden. Beide Möglichkeiten stellen Superkeys dar.

Allerdings brauchen Sie diese Spalten nicht, um eine Zeile zu identifizieren. Sie benötigen lediglich (als Beispiel) die *employeeID*. Diese stellt den minimalen Superkey dar, d.h. die minimale Anzahl Spalten, die zur Identifizierung einer einzelnen Zeile gebraucht wird. *employeeID* stellt also einen Schlüssel dar.

Schauen Sie sich noch einmal die Tabelle *employee* an. Einen Angestellten können wir über *name* oder *employeeID* identifizieren. Beide stellen Schlüssel dar. Wir bezeichnen sie als Schlüsselkandidaten, weil sie Kandidaten zur Bildung des Primärschlüssels sind. Der Primärschlüssel ist diejenige Spalte bzw. der Satz von Spalten, die wir zur Identifizierung einer einzelnen Zeile innerhalb einer Tabelle verwenden können. In unserem Fall werden wir daher *employeeID* zum Primärschlüssel erklären, was einen besseren Schlüssel als *name* ergibt, weil es Personen mit demselben Namen geben kann.

Fremdschlüssel stellen die Verknüpfungen zwischen Tabellen dar. Wenn Sie sich beispielsweise noch einmal die Abbildung 3.2 anschauen, werden Sie feststellen, dass die Spalte *departmentID* die Abteilungsnummer enthält. Dies ist der Fremdschlüssel. Die vollständigen Informationen über jede Abteilung werden in einer separaten Tabelle gespeichert, in der *departmentID* den Primärschlüssel bildet.

3.1.6 Funktionale Abhängigkeiten

Der Begriff *funktionale Abhängigkeiten* taucht weniger häufig als die zuvor genannten auf. Wir müssen uns aber trotzdem mit ihm beschäftigen, um den Normalisierungsprozess verstehen zu können, den wir weiter unten erörtern werden.

Wenn es eine funktionale Abhängigkeit zwischen der Spalte A und B in einer gegebenen Tabelle gibt (geschrieben als A → B), dann bestimmt der Wert der Spalte A den Wert der Spalte B. Beispielsweise bestimmt *employeeID* in der Tabelle *employee* funktional den Namen (*name*) (und auch alle anderen Attribute in diesem speziellen Beispiel).

3.1.7 Schema

Der Begriff *Schema* bzw. *Datenbankschema* meint einfach nur die Struktur oder den Entwurf der Datenbank, also den Aufbau der Datenbank ohne Daten darin. Das Schema ist sozusagen die Planungsgrundlage für die Daten in der Datenbank.

Wir können das Schema einer einzelnen Tabelle folgendermaßen beschreiben:

```
employee(employeeID, name, job, departmentID).
```

Wir werden in diesem Buch der Konvention folgen, Attribute mit durchgezogener Linie zu unterstreichen, die den Primärschlüssel repräsentieren, und alle anderen Attribute, die Fremdschlüssel repräsentieren, mit gestrichelter Linie zu unterstreichen. Primärschlüssel, die gleichzeitig Fremdschlüssel darstellen, werden sowohl eine durchgezogene als auch eine gestrichelte Linie aufweisen.

3.2 Prinzipien des Datenbankentwurfs

Beim Entwurf einer Datenbank müssen Sie zwei wichtige Dinge beachten:

- Welche Informationen müssen gespeichert werden? Das heißt also, was sind die Dinge bzw. Entitäten, über die wir Informationen speichern?
- Welche Fragen soll die Datenbank für uns beantworten? (Dies sind die so genannten Abfragen.)

Beim Nachdenken über diese Fragen müssen wir die Regeln des Vorgangs, den wir anfertigen möchten, im Hinterkopf behalten. Wir müssen uns also Gedanken über die Dinge machen, über die wir Daten speichern möchten, und darüber, wie die spezifischen Verknüpfungen dazwischen aussehen sollen.

Neben diesen Fragen müssen wir die Datenbank noch so strukturieren, dass strukturelle Probleme wie Redundanzen und Datenanomalien vermieden werden.

3.2.1 Redundanz versus Datenverlust

Beim Entwurf unseres Schemas müssen wir so vorgehen, dass wir die Redundanz der Daten minimieren und gleichzeitig Datenverluste vermeiden. Mit Redundanz sind Daten gemeint, die in verschiedenen Zeilen einer Tabelle oder unterschiedlichen Tabellen in der Datenbank wiederholt werden.

Stellen Sie sich dazu einmal vor, dass Sie anstelle der Tabellen *employee* und *department* nur eine einzige Tabelle namens *employeeDepartment* haben. Wir bauen diese Tabelle einfach dadurch auf, dass wir der Tabelle *employee* die Spalte *departmentName* hinzufügen. Das Schema würde dann folgendermaßen aussehen:

```
employeeDepartment(employeeID, name, job, departmentID, departmentName)
```

Für jeden Angestellten, der in der Abteilung Nr. 128, Research and Development, arbeitet, wiederholen wir die Daten »128, Research and Development« (wie die Abbildung 3.3 zeigt). Dasselbe gilt für alle anderen Abteilungen im Unternehmen.

employeeDepartment

employeeID	name	job	departmentID	departmentName
7513	Nora Edwards	Programmer	128	Research and Development
9842	Ben Smith	DBA	42	Finance
6651	Ajay Patel	Programmer	128	Research and Development
9006	Candy Burnett	Systems Administrator	128	Research and Development

Abbildung 3.3 Dieser Schemaentwurf führt dazu, dass der Abteilungsname (*departmentName*) wieder und wieder redundant gespeichert wird.

Dieses Konzept können Sie wie folgt abändern:

```
employee(employeeID, name, job, departmentID)

department(departmentID, name)
```

In diesem Fall wird jeder Abteilungsname nur einmal statt vielmals in der Datenbank gespeichert, was Speicherplatz spart und einige andere Probleme vermeidet.

Beachten Sie, dass wir *departmentID* in der Tabelle *employee* belassen müssen, weil wir andernfalls Informationen im Schema verlieren würden. In unserem Fall bedeutet das, dass wir die Verknüpfung zwischen einem Angestellten und der Abteilung, für die er arbeitet, aufgeben müssten. Beim Verbessern des Schemas müssen wir daher diese beiden Ziele, nämlich die Reduzierung von Datenwiederholungen in Verbindung mit dem Vermeiden von Datenverlust, stets im Hinterkopf behalten.

3.2.2 Datenanomalien

Datenanomalien stellen ein etwas komplexeres Problem dar. Datenanomalien treten beim Inhalt der Daten aufgrund eines Fehlers im Datenbankentwurf auf. Es gibt drei Arten von Anomalien, die auftreten können. Wir werden sie der Reihe nach anhand des fehlerhaften Schemas in Abbildung 3.3 behandeln.

Anomalien beim Einfügen

Anomalien beim Einfügen treten auf, wenn Sie Daten in eine fehlerhaft entworfene Tabelle einfügen. Stellen Sie sich dazu vor, dass im Unternehmen ein neuer Angestellter anfängt. Sobald wir die Daten zum neuen Angestellten in die Tabelle *employeeDepartment* einfügen, müssen wir sowohl die *departmentID* als auch den *departmentName* des Angestellten aufnehmen. Was geschieht nun, wenn wir Daten einfügen, die nicht mit dem übereinstimmen, was sich bereits in der Tabelle befindet? Nehmen Sie beispielsweise einmal an, Sie fügen einen Angestellten ein, der für Abteilung 42, Development, arbeitet. Es ist nun nicht mehr offensichtlich, welche Zeile in der Datenbank die richtige ist. Das ist eine Anomalie beim Einfügen.

Anomalien beim Löschen

Anomalien beim Löschen treten auf, wenn Sie Daten in einem fehlerhaften Schema löschen. Stellen Sie sich dazu vor, dass sämtliche Angestellte der Abteilung 128 aus irgendeinem Grund missgestimmt sind und am selben Tag kündigen. Wenn Sie alle diese Angestelltendatensätze löschen, besitzen Sie keinerlei Daten mehr über die Existenz von Abteilung 128 oder ihren Namen. Das ist eine Anomalie beim Löschen.

Anomalien beim Aktualisieren

Anomalien beim Aktualisieren treten auf, wenn Sie Daten in einem fehlerhaften Schema ändern. Stellen Sie sich dazu vor, dass sich die Abteilung 128 entschließt, ihren Namen in Emerging Technologies zu ändern. Nun müssen wir diese Daten für jeden Angestellten ändern, der für diese Abteilung arbeitet. Es ist klar, dass wir dabei mindestens einen Angestellten übersehen und einen Fehler machen … Und eben durch diesen einen (oder auch mehrere) Fehler entsteht eine Anomalie beim Aktualisieren.

3.2.3 Nullwerte

Eine zweckmäßige Regel für einen guten Datenbankentwurf lautet, dass wir einen Schemaentwurf vermeiden sollten, in dem eine große Anzahl leerer Attribute auftritt. Wenn wir beispielsweise festhalten möchten, dass bei einem von hundert Angestellten in unserer Firma eine spezielle Qualifikation vorliegt, würden wir zum Speichern dieser Information keine Spalte zur Tabelle *employee* hinzufügen, weil diese bei 99 Angestellten den Wert NULL enthielte. Anstelle dessen wäre es besser, eine neue Tabelle hinzuzufügen, in der nur *employeeIDs* und Qualifikationen für diejenigen Angestellten, die diese Qualifikationen auch besitzen, gespeichert würden.

3.3 Normalisierung

Die Normalisierung beschreibt einen Prozess, den wir zum Entfernen von Entwurfsfehlern aus einer Datenbank benutzen. Während der Normalisierung entwickeln wir eine Anzahl von normalen Formen, die aus Regeln bestehen, die genau beschreiben, was wir in unseren Tabellenstrukturen haben und nicht haben sollten. Der Normalisierungsprozess besteht aus dem Aufteilen der Tabellen in immer kleinere Tabellen, die zusammen einen besseren Entwurf ergeben.

Wir führen den Normalisierungsprozess durch, indem wir unseren Datenbankentwurf nach und nach in die verschiedenen Formen überführen. Grundsätzlich schließt jede Form die Formen darunter in sich ein. Damit sich ein Datenbankentwurf beispielsweise in der zweiten Normalform befinden kann, muss er sich ebenfalls in der ersten Normalform befinden. Damit sich ein Schema in der dritten Normalform befinden kann, muss es sich auch in der zweiten Normalform befinden usw. Auf jeder Stufe fügen wir weitere Regeln hinzu, die von unserem Schema erfüllt werden müssen.

3.3.1 Erste Normalform

Die erste Normalform, die manchmal auch 1NF genannt wird, legt fest, dass jedes Attribut oder jeder Spaltenwert atomisch sein muss. Das bedeutet, dass jedes Attribut nur einen einzigen Wert, aber keine Aufzählung von Werten oder eine andere Datenbankzeile, enthalten darf.

Die Abbildung 3.4 enthält eine Beispieltabelle.

employee

employeeID	name	job	departmentID	skill
7513	Nora Edwards	Programmer	128	C, Perl, Java
9842	Ben Smith	DBA	42	DB2
6651	Ajay Patel	Programmer	128	VB, Java
9006	Candy Burnett	Systems Administrator	128	NT, Linux

Abbildung 3.4 Dieser Schemaentwurf befindet sich nicht in der ersten Normalform, weil er eine Werteaufzählung in der Spalte *skill* enthält.

Hierbei handelt es sich um eine nicht normalisierte Version der Tabelle *employee*, die wir weiter oben erörtert haben. Wie Sie sehen, enthält sie eine zusätzliche Spalte namens *skill*, die die Qualifikationen jedes Angestellten beschreibt.

Jeder Wert in dieser Spalte besteht aus einer Aufzählung von Werten und ist daher kein atomischer Wert (wie etwa »Java«), weil er eine Liste von Werten wie »C, Perl, Java« usw. darstellt. Damit wird die Regel der ersten Normalform verletzt.

Um dieses Schema in die erste Normalform zu überführen, müssen Sie sämtliche Werte der Spalte *skill* in atomische Werte verwandeln. Es gibt eine Reihe von Möglichkeiten, um dieses Ziel zu erreichen. Die erste und vielleicht offensichtlichste zeigt die Abbildung 3.5.

employee

employeeID	Name	job	departmentID	skill
7513	Nora Edwards	Programmer	128	C
7513	Nora Edwards	Programmer	128	Perl
7513	Nora Edwards	Programmer	128	Java
9842	Ben Smith	DBA	42	DB2
6651	Ajay Patel	Programmer	128	VB
6651	Ajay Patel	Programmer	128	Java
9006	Candy Burnett	Systems Administrator	128	NT
9006	Candy Burnett	Systems Administrator	128	Linux

Abbildung 3.5 Alle Werte sind nun atomisch.

Pro Qualifikation des Angestellten (*skill*) haben wir eine Zeile gebildet. Dieses Schema befindet sich nun in der ersten Normalform. Aber auch diese Konstellation ist noch nicht die ideale, weil wir es immer noch mit einigen Redundanzen zu tun haben – zu jeder Kombination von *skill* und *employee* müssen wir sämtliche Angestelltendetails speichern.

Eine bessere Lösung und die richtige Methode, um diese Daten in die erste Normalform zu überführen, zeigt die Abbildung 3.6.

In diesem Beispiel haben wir *skill* getrennt und in einer separaten Tabelle untergebracht, die nur noch *employeeID* und *skill* miteinander verknüpft. Damit sind wir das Problem der Redundanz los.

Sie fragen sich vielleicht, woher wir wissen, dass wir bei der zweiten Lösung angekommen sind. Darauf gibt es zwei Antworten: Die eine ist die Erfahrung. Die zweite lautet, dass wir beim Schema in der Abbildung 3.6 ankommen werden, wenn wir das Schema in der Abbildung 3.5 nehmen und mit dem Normalisierungsprozess fortfahren. Der Vorteil der Erfahrung ermöglicht es uns, einfach direkt zu diesem Entwurf zu gehen. Es ist aber ebenso vollkommen gültig, den Normalisierungsprozess bis zum Ende durchzuführen.

employee

employeeID	name	job	departmentID
7513	Nora Edwards	Programmer	128
9842	Ben Smith	DBA	42
6651	Ajay Patel	Programmer	128
9006	Candy Burnett	Systems Administrator	128

employeeSkills

employeeID	skill
7513	C
7513	Perl
7513	Java
9842	DB2
6651	VB
6651	Java
9006	NT
9006	Linux

Abbildung 3.6 Wir haben das Problem durch Erstellen einer zweiten Tabelle gelöst.

3.3.2 Zweite Normalform

Nachdem wir ein Schema in die erste Normalform überführt haben, können wir zu den höheren Formen fortschreiten, die etwas schwieriger zu verstehen sind.

Ein Schema befindet sich dann in der zweiten Normalform (auch als 2NF bezeichnet), wenn sämtliche Attribute, die kein Bestandteil des Primärschlüssels sind, funktional voll abhängig vom Primärschlüssel sind und sich das Schema bereits in der ersten Normalform befindet. Was heißt das? Das bedeutet, dass jedes Nicht-Schlüssel-Attribut funktional abhängig von sämtlichen Schlüsselbestandteilen sein muss. Wenn der Primärschlüssel aus mehreren Spalten besteht, müssen alle anderen Attribute in der Tabelle *von der Kombination dieser Werte* abhängig sein.

Schauen Sie sich dazu ein Beispiel an, das die Dinge klarer macht.

Die Abbildung 3.5 enthält das Schema mit der einzelnen Zeile pro *skill* in der Tabelle *employee*. Dieser Schemaentwurf befindet sich in der ersten Normalform, aber nicht in der zweiten. Weshalb nicht? Was ist der Primärschlüssel dieser Tabelle? Uns ist bekannt, dass der Primärschlüssel eine einzelne Zeile in der Tabelle eindeutig identifizieren können muss. In diesem Fall besteht die einzige Möglichkeit dazu in der Verwendung einer Kombination aus *employeeID* und *skill*. Mit der Art und Weise, wie *skill* eingerichtet ist, kann mit *employeeID* allein keine eindeutige Identifizierung einer Zeile vorgenommen werden. Die *employeeID* 7513 beispielsweise identifiziert gleich 3 Zeilen. Jedoch die Kombination aus *employeeID* und *skill* würde eine einzige Zeile identifizieren können; wir verwenden daher diese beiden in Kombination als unseren Primärschlüssel. Damit erhalten wir das folgende Schema:

```
employee(employeeID, name, job, departmentID, skill)
```

Als Nächstes müssen wir die Frage stellen: Worin bestehen hier die funktionalen Abhängigkeiten? Wir haben nun:

```
employeeID, skill -> name, job, departmentID
```

und haben ebenso:

```
employeeID -> name, job, departmentID
```

Anders ausgedrückt: Aus *employeeID* allein können wir jetzt *name*, *job* und *departmentID* bestimmen. Das bedeutet, dass diese Attribute nicht vollständig, sondern teilweise funktional abhängig vom Primärschlüssel sind. Sie können also diese Attribute aus einem Teil des Primärschlüssels bestimmen, ohne dazu den gesamten Primärschlüssel zu benötigen. Damit befindet sich dieses Schema also nicht in der zweiten Normalform.

Die nächste Frage lautet: Wie können wir es in die zweite Normalform überführen?

Dazu müssen wir die Tabelle in Tabellen zerlegen, in denen sämtliche Nicht-Schlüssel-Attribute funktional vollständig abhängig vom Schlüssel sind. Es ist offensichtlich, dass wir dieses Ziel durch Aufteilung der Tabelle in zwei Tabellen erreichen:

```
employee(employeeID, name, job, departmentID)

employeeSkills(employeeID, skill)
```

Dies ist das Schema, das wir bereits in Abbildung 3.6 hatten.

Wie schon erörtert, befindet sich dieses Schema in der ersten Normalform, weil sämtliche Werte nun atomisch sind. Es befindet sich außerdem in der zweiten Normalform, weil jedes Nicht-Schlüssel-Attribut jetzt funktional vollständig abhängig von sämtlichen Teilen des Schlüssels ist.

3.3.3 Dritte Normalform

Vielleicht haben Sie schon einmal sagen hören: »Bei der Normalisierung geht es um den Schlüssel, um den ganzen Schlüssel und um nichts anderes als den Schlüssel«. Die zweite Normalform teilt uns mit, dass die Attribute vom gesamten Schlüssel abhängen müssen. Die dritte Normalform teilt uns mit, dass die Attribute von nichts anderem als dem Schlüssel abhängen dürfen.

Damit sich ein Schema in der dritten Normalform (3NF) befindet, müssen wir (formal betrachtet) sämtliche transitiven Abhängigkeiten entfernen. Außerdem muss sich das Schema bereits in der zweiten Normalform befinden. Und was ist nun eine transitive Abhängigkeit?

Schauen Sie sich dazu noch einmal die Abbildung 3.3 mit dem folgenden Schema an:

```
employeeDepartment(employeeID, name, job, departmentID, departmentName)
```

Dieses Schema enthält die folgenden funktionalen Abhängigkeiten:

```
employeeID -> name, job, departmentID, departmentName

departmentID -> departmentName
```

Der Primärschlüssel ist *employeeID*, während alle Attribute funktional vollständig abhängig von diesem sind. Es ist nicht besonders schwer zu entdecken, weil es im Primärschlüssel nur ein Attribut gibt!

Sie können Folgendes erkennen:

```
employeeID -> departmentName
```

```
employeeID -> departmentID
```

Und außerdem:

```
departmentID -> departmentName
```

Beachten Sie außerdem, dass das Attribut *departmentID* kein Schlüssel ist.

Diese Relation bedeutet, dass die funktionale Abhängigkeit `employeeID -> departmentName` eine transitive Abhängigkeit darstellt. Tatsächlich liegt noch ein Zwischenschritt vor (die Abhängigkeit `departmentID -> departmentName`).

Um zur dritten Normalform zu gelangen, müssen wir diese transitive Abhängigkeit beseitigen.

Wie bei den Normalformen zuvor zerlegen wir diese Tabelle zur Überführung in die dritte Normalform in weitere Tabellen. Auch in diesem Fall ist recht eindeutig, wie wir dies machen müssen. Wir fügen dem Schema zwei Tabellen *employee* und *department* hinzu, etwa so:

```
employee(employeeID, name, job, departmentID)
```

```
department(departmentID, departmentName)
```

Damit kommen wir zu dem Schema für *employee* zurück, mit dem wir in Abbildung 3.2 begonnen haben. Dieses Schema befindet sich nun in der dritten Normalform.

Eine andere Beschreibung der dritten Normalform lautet, dass für ein Schema, welches sich in der dritten Normalform befindet, für jede funktionale Abhängigkeit in allen Tabellen gilt, dass

- die linke Seite der funktionalen Abhängigkeit ein Superkey ist (also ein Schlüssel, der nicht notwendigerweise minimal ist),

oder

- die rechte Seite der funktionalen Abhängigkeit Teil eines Schlüssels dieser Tabelle ist.

Der zweite Teil kommt allerdings nicht besonders häufig vor … In den meisten Fällen werden alle funktionalen Abhängigkeiten von der ersten Regel abgedeckt.

3.3.4 Boyce-Codd-Normalform

Die letzte Normalform, die wir kurz betrachten, ist die Boyce-Codd-Normalform, die manchmal auch BCNF genannt wird. Sie stellt eine Variante der dritten Normalform dar. Gewiss erinnern Sie sich noch an die beiden zuvor erwähnten Regeln. Damit sich eine Relation in BCNF befindet, muss sie sich in der dritten Normalform befinden und unter die erste der beiden Regeln fallen. Das bedeutet, dass sämtliche funktionalen Abhängigkeiten über einen Superkey auf der linken Seite verfügen müssen.

Dies ist wie in unserem Beispiel meistens der Fall, ohne dass wir dazu besondere Schritte unternehmen müssen. Falls wir eine Abhängigkeit haben, die diese Regel bricht, müssen wir wieder wie beim Überführen in 1NF, 2NF und 3NF die Daten zerlegen.

3.3.5 Höhere Normalformen

Es existieren noch höhere Normalformen (vierte, fünfte usw.), die jedoch hauptsächlich akademisch interessant sind und den Datenbankentwurf in der Praxis weniger betreffen. Die 3NF (bzw. BCNF) genügt zur Beseitigung der Datenredundanzen, die Ihnen in der Praxis begegnen werden.

3.4 Zusammenfassung

Als Zusammenfassung nun eine Übersicht der in diesem Kapitel behandelten Themen.

3.4.1 Konzepte

- Entitäten sind Dinge aus der Praxis, die miteinander verknüpft sein können.
- Relationen bzw. Tabellen speichern Daten in tabellarischer Form.
- Die Spalten in den Tabellen beschreiben die Attribute, die zu jedem Datum gehören.
- Die Zeilen in Tabellen enthalten Daten mit Werten in jeder Spalte der Tabelle.
- Schlüssel werden zur Identifizierung einer einzelnen Zeile verwendet.
- Funktionale Abhängigkeiten legen fest, welche Attribute die Werte anderer Attribute bestimmen.
- Ein Schema stellt die Planungsgrundlage einer Datenbank dar.

3.4.2 Entwurfsprinzipien

- Redundanzen ohne Verlust an Daten minimieren.
- Anomalien beim Einfügen, Löschen und Aktualisieren sind Probleme, die beim Einfügen, Löschen oder Aktualisieren von Daten in Tabellen mit fehlerhafter Struktur auftreten.
- Entwürfe vermeiden, die zu großen Mengen an Nullwerten führen.

3.4.3 Normalisierung

- Die Normalisierung bezeichnet den formalen Prozess der Verbesserung eines Datenbankentwurfs.
- Die erste Normalform (1NF) legt atomische Spalten bzw. Attributwerte fest.
- Die zweite Normalform (2NF) bedeutet, dass die Attribute außerhalb des Schlüssels vom gesamten Schlüssel abhängen müssen.
- Die dritte Normalform (3NF) bedeutet, dass keine transitiven Abhängigkeiten vorliegen dürfen.
- Die Boyce-Codd-Normalform (BCNF) bedeutet, dass sämtliche Attribute funktional von einem Superkey bestimmt sein müssen.

3.5 Quiz

1. Ein Superkey ist:
   a. ein minimaler Schlüssel.
   b. ein Fremdschlüssel.
   c. ein Satz von Attributen, der zur Identifizierung einer Zeile in der Tabelle verwendet werden kann.
   d. ein minimaler Satz von Attributen, der zur Identifizierung einer Zeile in der Tabelle verwendet werden kann.
2. Wenn sich eine Tabelle in der ersten Normalform befindet, dann:
   a. befindet sie sich ebenfalls in der ersten Normalform.
   b. befindet sie sich ebenfalls in der dritten Normalform.
   c. enthält sie keinerlei transitive Abhängigkeiten.
   d. enthält sie Attribute, die nicht vollständig funktional vom Schlüssel abhängen.
3. Wenn sich eine Tabelle in der dritten Normalform befindet, dann:
   a. befindet sie sich ebenfalls in der Boyce-Codd-Normalform.
   b. enthält sie keine nicht atomischen Attribute.
   c. enthält sie keinerlei transitive Abhängigkeiten.
   d. enthält sie Attribute, die nicht vollständig funktional vom Schlüssel abhängen.
4. Die drei Arten von Anomalien sind:
   a. Einfügen, Selektieren, Löschen
   b. Einfügen, Aktualisieren, Löschen
   c. Selektieren, Aktualisieren, Löschen
5. Ein Tupel ist:
   a. eine Spalte
   b. eine Zeile
   c. ein Schlüsselkandidat
   d. der Geburtsort von Elvis Presley
   e. ein Fremdschlüssel

3.6 Übungen

1. Überführen Sie das folgende Schema in die dritte Normalform:

   Customers(customerID, customerName, customerAddress orderID, orderDate, itemID, itemName, itemQuantity)
2. Versuchen Sie ein Schema zu entwerfen, das sich in 3NF, aber nicht in BCNE befindet.

3.7 Antworten

3.7.1 Quiz

1. C
2. A
3. C
4. B
5. B (Elvis wurde in Tupelo geboren.)

3.7.2 Übungen

1.

   ```
   Customers(customerID, customerName, customerAddress)

   Orders(orderID, orderDate, customerID)

   OrderItems(orderID, itemID, itemQuantity)

   Items(itemID, itemName)
   ```

2. Es gibt viele mögliche Antworten – achten Sie einfach nur darauf, dass Ihre Antwort mit den entsprechenden Normalisierungsregeln übereinstimmt.

3.8 Als Nächstes

In Kapitel 4, *Datenbanken, Tabellen und Indizes erstellen*, werden wir ein Datenbankschema in eine echte MySQL-Datenbank verwandeln.