1 Architektur, Modellierung und Entwurf

Dieses erste Kapitel richtet sich an Datenbankneulinge und an alle, denen die Grundlagen relationaler Datenbanksysteme, die Techniken zur Datenmodellierung oder die Normalisierung nicht vertraut sind. Zu Beginn wird kurz auf das Konzept der relationalen Datenbanken und auf die Architektur relationaler Systeme eingegangen. Daran anschließend wird mit den Mitteln der Entity-Relationship-Modelle (ERM) ein Datenmodell entwickelt und danach in einen Datenbankentwurf transformiert. Der letzte Abschnitt gilt der Normalisierung, einem wichtigen Instrument, um stabile, widerspruchs- und redundanzfreie Datenbankstrukturen zu entwickeln. Sie lernen, wie die Normalisierungsregeln angewendet werden und dass Redundanz manchmal sinnvoll ist und zugelassen werden kann. Nach diesem Kapitel sollten Sie in der Lage sein, ein Datenmodell zu entwerfen und in einen normalisierten Datenbankentwurf zu überführen.

Modellbildung und Normalisierung helfen Ihnen bereits in der Entwurfsphase, logische Unstimmigkeiten zu finden und dadurch die Änderungsanfälligkeit einer Anwendung zu minimieren. Besonders bei großen Projekten ist die Modellierung der Daten empfehlenswert, sie hilft, Ihr Projekt überschaubar zu machen und die Komplexität zu beherrschen. Wenn Sie die Grundlagen von Datenbanksystemen kennen oder diesen Abschnitt auf später verschieben möchten, geht es im zweiten Kapitel mit der Installation des Datenbanksystems PostgreSQL weiter und im dritten Kapitel mit seiner Anwendung.

1.1 Datenbanken – was ist das?

Auf diese einfache Frage gibt es keine einfache Antwort, aber eine lange Liste von Büchern zum Thema. Darum werde ich mich auf eine kurze Darstellung beschränken.

Datenbanken sind komplex strukturierte Sammlungen von Daten, die dauerhaft gespeichert sind und von vielen Anwendern oder Anwendungsprogrammen, auch gleichzeitig, als gemeinsame Informationsbasis genutzt werden können. Sie bieten die Möglichkeit, große Datenbestände effizient zu verwalten und flexibel abzufragen. Nicht nur die Daten als solche können in einer Datenbank gespeichert werden, sondern auch Beziehungen der Daten untereinander, so dass bei einer Abfrage diese Daten miteinander verknüpft und zugeordnet werden können. Diese Eigenschaft macht Datenbanken zu einem mächtigen Instrument zur Speicherung und Gewinnung von Informationen.

Relationale Datenbanken

Es gibt verschiedene Datenbanktechnologien wie hierarchische oder objektorientierte Datenbanksysteme. Da dieses Buch sich mit PostgreSQL, einem objektrelationalen Datenbanksystem, beschäftigt, werde ich die anderen Technologien außer Betracht lassen. Objektrelational bedeutet, dass ein relationales System um Eigenschaften objektorientierter Systeme erweitert wurde. In diesem Kapitel geht es um die Eigenschaften relationaler Datenbanken. Auf die objektrelationalen Erweiterungen von PostgreSQL wird später eingegangen.

Die Struktur einer relationalen Datenbank ist ein Geflecht von Tabellen, in denen jeweils gleich strukturierte Daten in Spalten angeordnet sind. Die Inhalte in den einzelnen Spalten können so unterschiedlich sein wie die Anwendungen, die auf diesen Daten operieren, sie haben aber alle eines gemeinsam: Es sind Modelle von Ausschnitten aus der realen Welt, die, abstrahiert auf ihre anwendungsrelevanten Eigenschaften, in den Tabellen gespeichert werden, unabhängig davon, wie die Daten von den Anwendungen oder Benutzern gebraucht werden. (Ein einfaches Beispiel einer Tabelle sehen Sie in Abb. 1–8 in diesem Kapitel.)

Modellierung der realen Welt

Jede Person, jedes Ding der realen Welt ist durch ihre/seine Eigenschaften identifizierbar und in unterschiedliche Zusammenhänge eingebunden. So ist ein Buch durch seine ISBN, ein Auto durch sein Kennzeichen und eine Webseite durch ihre Internetadresse eindeutig identifizierbar. Solche Abstraktionen von Ausschnitten aus der realen Welt werden in der Datenbankterminologie Entität (entity) genannt. Modellbildung bedeutet, dass man alle Eigenschaften (Attribute oder properties) und Beziehungen einer Entität, die für die geplante Anwendung notwendig sind, auf ein Datenmodell abbildet, das dann in der Datenbank gespeichert wird.

Wartungs- und Änderungsfreundlichkeit

In einer Datenbank werden die Daten nur einmal definiert und zentral für alle Benutzer verwaltet. (Der Einfachheit halber werde ich nicht mehr zwischen Benutzern und Anwendungsprogrammen unterscheiden, da beide auf dieselbe Weise auf die Daten zugreifen.) Bereits hier wird ein wichtiger Vorteil des Datenbankkonzepts offensichtlich: die Wartungs- und Änderungsfreundlichkeit. Änderungen müssen nur an einer Stelle durchgeführt werden und stehen danach allen Benutzern zur Verfügung. Wenn sich in einem Online-Shop die Preise für einen Artikel ändern, muss nur der dazugehörige Eintrag in der Datenbank geändert werden. Beim nächsten Aufruf der Seite wird der aktualisierte Wert aus der Datenbank ausgelesen. Bei statischen HTML-Seiten müsste jede Seite, auf der der Preis dieses Artikels steht, editiert werden.

Vermeidung von Redundanz

Weil Daten nur einmal gespeichert werden und dadurch Redundanz vermieden wird, ist es möglich, den Datenbestand konsistent zu halten, so dass alle Benutzer mit denselben Daten arbeiten. Für eine gemeinsame Informationsbasis ist diese Voraussetzung unerlässlich.

1.2 Datenbankmanagementsystem

Das DBMS kontrolliert die Datenbank

Die Verwaltung des gesamten Datenbestands übernimmt das Datenbankmanagementsystem, DBMS. Zusammen mit der Datenbank, in der die eigentlichen Daten gespeichert werden, bildet es das Datenbanksystem. Alle Zugriffe auf Daten erfolgen ausschließlich über das DBMS. Über dieses Managementsystem werden Tabellen definiert oder Daten modifiziert, es sorgt für die korrekte Speicherung der Daten und bearbeitet Anfragen der Benutzer. Es ist die Schnittstelle zwischen Anwendern und Datenhaltung, so dass ein Benutzer sich nicht um Implementierungsdetails oder um die Speicherung von Daten kümmern muss und auch nicht von Änderungen der Implementierung oder der internen Organisation der Daten betroffen ist.

Physische Datenunabhängigkeit

Diese Trennung von Benutzern und Daten ist als physische Datenunabhängigkeit bekannt. Dies vereinfacht den Umgang mit dem System, weil der Benutzer nur seine Daten und ihre Eigenschaften kennen muss und nicht deren interne Organisation. Er schickt einen Auftrag an das DBMS und erhält die Ergebnisse vom DBMS zurück. Ihn interessiert nicht das »Wie«, sondern das »Was«. Datenbanksysteme, bei denen über ein Datenbankmanagementsystem auf die Daten zugegriffen wird, werden allgemein als Client-Server-Systeme bezeichnet, wobei es unerheblich ist, ob Client und Server auf derselben Maschine laufen oder ob der Client über eine Internetverbindung auf den Datenbankserver zugreift.

Abb. 1–1
Architektur eines Datenbanksystems

1.3 Architektur eines Datenbanksystems

Der Dreiteilung »Anwender – DBMS – physikalische Datenorganisation« entspricht die Architektur von Datenbanksystemen, die in drei Ebenen aufgebaut ist:

- die externe Ebene, das, was die Benutzer »sehen«,
- die konzeptuelle Ebene, also die logische Gesamtsicht, und die
- interne Ebene, die Datenorganisation auf den Speichern.

Data Definition Language

Auf jeder dieser Ebenen werden die Daten auf einem entsprechenden Abstraktionsniveau modelliert, zwischen den Ebenen sorgen Transformationsregeln für die korrekte Übergabe. Die Beschreibungen dieser Modelle werden in einer Datendefinitionssprache (DDL, Data Definition Language) festgelegt und heißen Schemata.

Entsprechend unterscheidet man die verschiedenen externen Schemata, ein konzeptuelles Schema und ein internes Schema. Ein Schema ist eine Art Vorlage, von der Ausprägungen erzeugt werden können.
Bei der Entwicklung von Webanwendungen haben wir es mit der externen und der konzeptuellen Ebene zu tun. Die interne Ebene, also die Organisation der Daten auf den Speichern, wird vom Datenbanksystem, in unserem Fall PostgreSQL, vorgegeben.

1.3.1 Die konzeptuelle Ebene

Logische Gesamtsicht

Die konzeptuelle Ebene repräsentiert die Gesamtheit der Daten und ihrer Beziehungen untereinander, unabhängig davon, wie die Benutzer die Daten brauchen. Sie wird auch als logische Gesamtsicht bezeichnet. Sie ist der Bezugspunkt sowohl für alle Anwendungen der externen Ebene als auch für die Abbildung der Daten auf die Speicher der internen Ebene. Auf der konzeptuellen Ebene werden alle Informationseinheiten und ihre Beziehungen beschrieben. Hier wird der Realweltausschnitt erfasst, in einem Datenmodell formuliert und in eine Datenbankstruktur umgesetzt. Wenn Ihre Anwendung die Registrierung von Benutzern vorsieht, so müssen für einen Benutzer die Eigenschaften »Benutzername« und »Kennwort« in das Modell aufgenommen und in die Logik der Anwendung integriert werden.

Integritätsbedingungen

Außerdem können Bedingungen formuliert werden, denen die Daten genügen müssen, so genannte Integritätsbedingungen**.** Beispielsweise kann definiert werden, dass bestimmte Daten erst nach einer festgelegten Zeitspanne gelöscht werden können oder dass Preisangaben keine negativen Werte annehmen dürfen.

Die Beschreibung der konzeptuellen Ebene sollte sich nicht an der technischen Umsetzung, sondern an den logischen Zusammenhängen der Informationen orientieren. Es ist gute Praxis, Personen, die mit den Anwendungen und Daten später arbeiten, diese logische Gesamtsicht entwickeln zu lassen. Sie kennen den Kontext genau und können ein relativ stabiles Modell ihrer Arbeitswelt entwerfen, das um so weniger änderungsbedürftig ist, je sorgfältiger es geplant wurde. Spätere Änderungen des konzeptuellen Schemas ziehen oft weitreichende Modifikationen nach sich.

1.3.2 Die externe Ebene

Verschiedene Sichten auf denselben Datenbestand

Die Benutzer einer Datenbankanwendung »sehen« in der Regel nur den Ausschnitt aus dem konzeptionellen Schema, der für ihre Aufgabenstellung relevant ist. Deshalb wird dieser Ausschnitt auch als Sicht (view) bezeichnet. So können viele verschiedene Benutzer genauso viele verschiedene Sichten auf denselben Datenbestand haben. Stellen Sie sich einen Onlinekatalog vor, bei dem ein Benutzer in den Büchern sucht, ein anderer im Antiquariat stöbert, während ein Mitarbeiter des Shops sich gerade den aktuellen Lagerbestand anzeigen lässt. Jeder der drei hat seine eigene Sicht auf denselben Datenbestand.

DML – Data Manipulation Language

Auf der externen Ebene müssen die Benutzer Sprachmittel oder Werkzeuge zur Hand haben, um auf die Daten in der Datenbank

zugreifen zu können. Der Administrator des Onlinekatalogs muss Artikeldaten aktualisieren können und der Besucher der Webseite muss Daten auswählen können. Änderungen oder Abfragen von Daten werden mittels einer Datenmanipulationssprache (DML, Data Manipulation Language) an das DBMS geschickt, welches die angeforderten Daten von den Speichern liest und an den Benutzer zurückgibt.

SQL – die Standardsprache

Bei relationalen Systemen ist SQL (Structured Query Language) die Standardsprache zum Zugriff auf Datenbanken und auch zur Datendefinition. Alle SQL-Anweisungen werden offiziell »statement« genannt. In der Praxis und auch in der Literatur wird ganz allgemein von »querys« (Abfragen) gesprochen, obwohl diese ganz unterschiedliche Aktionen bewirken können, beispielsweise Daten definieren, lesen oder modifizieren.

Datenbankterminals

Zu den Werkzeugen, die die meisten Datenbanksysteme zur Verfügung stellen, gehören so genannte Monitore oder Datenbankterminals. Das sind kommandozeilenorientierte Clients, mit denen direkt SQL-Kommandos an den Datenbankserver geschickt und dessen Rückgaben angezeigt werden können. Außerdem stellt das System dem Administrator eine ganze Reihe von Werkzeugen zur Verfügung, mit denen er Daten reorganisieren oder Benutzer verwalten kann. Für die Benutzer unseres Onlinekatalogs sind diese Werkzeuge natürlich nicht verfügbar, es wäre weder wünschenswert noch sinnvoll, wenn die Benutzer die darunter liegende Datenbank unkontrolliert manipulieren könnten, sondern verheerend. Bei datenbankgestützten Internetseiten wird deshalb mit einem Skript eine Benutzerschnittstelle implementiert, die die Zugriffe auf die Datenbank kontrolliert. Das Skript schickt SQL-Befehle über den Webserver an das DBMS, umgekehrt nimmt der Webserver die Ausgaben vom DBMS entgegen und gibt sie auf der Webseite aus (siehe Abbildung 1–1).

1.4 Datenmodellierung

Nachdem mehrfach die Rede von Daten und ihren Beziehungen war, wollen wir nun Datenmodelle entwickeln. Dieser Prozess läuft in zwei Stufen ab: Zuerst wird ein semantischer Entwurf des konzeptuellen Schemas erstellt, der dann in der zweiten Phase mittels der Datendefinitionssprache auf eine Struktur abgebildet wird, die den semantischen Entwurf widerspiegelt und vom DBMS verstanden wird.

Entity-Relationship-Modell

Es gibt mehrere Verfahren zur Datenmodellierung (semantische Verfahren oder funktionale Verfahren), als De-facto-Standard hat sich das Entity-Relationship-Modell etabliert, das 1976 zum ersten Mal vorgestellt und seither mehrfach erweitert wurde.

Wie weiter oben erwähnt, ist eine Entität ein »Etwas«, das eindeutig beschreibbar ist. Entitäten können Objekte sein, wie etwa ein Buch, ein Zwanzig-Euro-Schein oder ein Auto, aber auch Ereignisse, wie ein Konzert oder ein Arztbesuch.

Für die Darstellung der Entitäten gibt es im ER-Modell drei Basiskonstrukte:

- Entitäten, die die Objekte darstellen, die modelliert werden sollen,
- Attribute oder Eigenschaften, mit denen die Objekte beschrieben werden, und
- Beziehungen, über die Entitäten miteinander verknüpft werden können.

Die grafische Darstellung dieser Konstrukte ist in der Literatur uneinheitlich. Entitätstypen werden meistens als Rechtecke, Attribute als Kreise oder Ellipsen und Beziehungen entweder als Rauten, mit Kanten zu den beteiligten Entitäten, oder nur als Kanten dargestellt.

1.4.1 Entitäten und Attribute

Entitäten werden nicht als individuelle Objekte aufgefasst, sondern als Typen von Objekten, von denen es beliebig viele Ausprägungen geben kann. Der Begriff Entität hat eine doppelte Bedeutung, er bezeichnet diesen Typ und repräsentiert gleichzeitig die Menge aller Ausprägungen dieses Typs.

Eine Entität wird durch die Menge ihrer Attribute beschrieben.

Eine Entität wird durch die Menge ihrer Attribute beschrieben. Ein Buch in einem Onlineshop kann durch mehrere Attribute eindeutig beschrieben werden. Wir fügen der Entität `Buch` die Attribute zu, die für die geplante Anwendung notwendig sind, und erzeugen dadurch ein Muster, eine Schablone, für alle Ausprägungen der Entitäten vom Typ `Buch`.

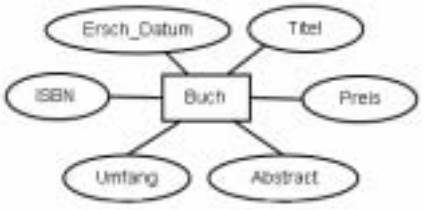


Abb. 1–2
Die Entität `Buch` *mit Attributen*

Wertebereiche von Attributen

Jedes Attribut kann nur Werte aus einem bestimmten Wertebereich annehmen. Im obigen Beispiel ist der Wertebereich für das Attribut `Preis` die Menge der rationalen Zahlen. Der Wertebereich für das Attribut `Umfang` umfasst alle positiven Integerzahlen.

Attributwerte müssen unteilbar sein.

Attributwerte dürfen nicht weiter zerlegbar und auch keine Ergebnisse von Berechnungen sein. Unzulässig wäre beispielsweise ein Attribut `NameDesAutors`, das in Vorname und Nachname aufgeteilt werden kann, oder ein Attribut, das den Betrag für die Mehrwertsteuer eines Buchs enthält. Ändert sich der Buchpreis oder der Steuersatz, so werden alle Werte des Attributs falsch.

Indem man den Attributen Werte zuweist, erzeugt man von diesem Muster Ausprägungen. Über die Kombination aller Attributwerte lassen sich die Ausprägungen eindeutig identifizieren. Oft genügt zur Identifikation eine Kombination weniger Attributwerte oder ein einzelnes Attribut. So wird ein Buch allein durch das Attribut `ISBN` eindeutig identifiziert. Die Abbildung einer Entität im relationalen System ist denkbar einfach, die Ausprägungen werden in Tabellen gehalten, in denen jedes Attribut eine Spalte beschreibt und jede Ausprägung eine Zeile (oder einen Datensatz) darstellt.

Logische Datenunabhängigkeit

Es hängt von der Aufgabenstellung ab, mit welchen Attributen eine Entität beschrieben wird, das heißt, welche Attribute in das Datenmodell aufgenommen werden. Das Attribut `Abstract` ist zur Identifikation eines Datensatzes vom Typ `Buch` nicht notwendig, wohl aber für die Präsentation auf der Webseite, die dem Besucher diesen Text anzeigt. Das Datenmodell soll die Attribute enthalten, die von der geplanten Anwendung benötigt werden. Wenn neue Anwendungen dazukommen, kann man neue Attribute hinzufügen, ohne dass dadurch schon bestehende Anwendungen betroffen sind. Durch die Entkoppelung von Daten und Anwendungen in der Architektur wird die logische Datenunabhängigkeit gewährleistet, so dass man das konzeptuelle Schema modifizieren kann, ohne dass die externen Schemata geändert werden müssen.

1.4.2 Subtypen und Supertypen

Jeder Untertyp ist zugleich eine Instanz des Supertyps

Oft gibt es Entitäten, die in ihrer Struktur einige gemeinsame Attribute haben. Alle Artikel des Onlineshops haben einen Preis und einen Artikelnamen als gemeinsame Attribute. Bücher aus dem Onlineshop haben außerdem als spezifisches Attribut eine ISBN und eine Seitenzahl, während ein Videofilm die Lauflänge des Films als Attribut haben kann. Die gemeinsamen Attribute kann man zu einem Supertyp zusammenfassen, der dann eine Generalisierung der verschiedenen Untertypen (Subtypen) darstellt, wenn jede Ausprägung des Supertyps genau eine Ausprägung eines Untertyps ist. Die Untertypen haben neben den gemeinsamen Attributen mindestens ein eigenes Attribut und können somit als Spezialisierungen des Supertyps aufgefasst wer-

den. Jeder Untertyp ist zugleich eine Instanz des Supertyps. Wesentlich ist, dass die Untertypen, neben ihren eigenen spezifischen Attributen, alle Attribute des Supertyps haben, aber nicht notwendigerweise seine Beziehungen. Die Unterstützung von Generalisierung bzw. Spezialisierung ist ein Merkmal objektrelationaler Datenbanksysteme.

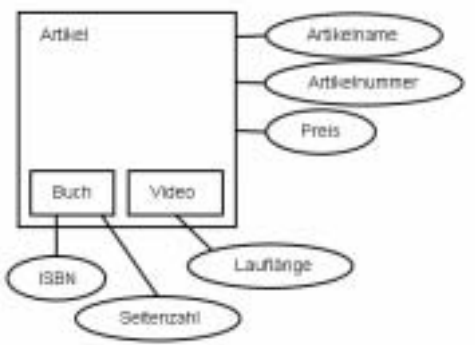


Abb. 1–3
Superentität Artikel, Subentitäten Buch/Video

1.4.3 Beziehungen

Entitäten sind Abstraktionen der Realwelt, und so, wie Personen und Dinge in ihre Zusammenhänge eingebunden sind, können Entitäten zu anderen Entitäten in Beziehung stehen. Eine Entität `Buch` steht in der Beziehung `wurde geschrieben von` zu einer Entität `Person`, oder in der Beziehung `wird geliefert von` zu einer Entität `Lieferant`.

Beziehungstypen

Genauso wie bei den Entitäten werden nicht einzelne Beziehungen zwischen einzelnen Entitäten betrachtet, sondern der Beziehungstyp. Dann bezeichnet der Typ `wird geliefert von` eine Beziehung zwischen dem Entitätstyp `Buch` und dem Entitätstyp `Lieferant` und zugleich die Menge aller dieser Beziehungen. Beziehungen zwischen Entitäten sind nicht gerichtet, sie gelten in beide Richtungen. Wenn zwischen den Entitäten `Buch` und `Lieferant` die Beziehung `B` gilt, so umfasst `B` für alle Ausprägungen von `Buch` die zugehörigen Ausprägungen von `Lieferant` und umgekehrt.

Beziehungen sind Zuordnungen und werden aufgrund ihrer Kardinalität eingeteilt:

1:1-Beziehung (one to one): Das ist die einfachste Form einer Beziehung. Jeder Ausprägung von `E1` ist höchstens eine Ausprägung von `E2` zugeordnet und umgekehrt. Ein typischer Vertreter dieser Kategorie: `ist verheiratet mit` (dies ist in unserem Kulturkreis eine `1:1`-Beziehung). Oder ein Sachbearbeiter ist für genau einen Bereich allein

zuständig und kann dann im Gegenzug über den Bereich eindeutig identifiziert werden.

Abb. 1–4
Eine 1:1-Beziehung

1:n-Beziehung (one to many): Jeder Ausprägung von `E1` sind beliebig viele, oder auch keine, Ausprägungen von `E2` zugeordnet, und jeder Ausprägung von `E2` ist höchstens eine Ausprägung von `E1` zugeordnet. Ein Beispiel hierfür ist der Verlag, der zwar mehrere Bücher verlegen kann, aber ein Buch wird immer nur von genau einem Verlag herausgegeben. Von einer Entität `Buch` kann über die Beziehung `verlegt` der Verlag ermittelt werden. Der Sachbearbeiter ist jetzt für mehrere Bereiche zuständig, die nur er bearbeitet. Wenn man einen »seiner« Bereiche kennt, kann man ihn über die Beziehung identifizieren.

Abb. 1–5
Eine 1:n-Beziehung

m:n-Beziehung (many to many): Jeder Ausprägung von `E1` können mehrere Ausprägungen von `E2` zugeordnet sein und umgekehrt. Beispiel: Ein Autor kann mehrere Bücher verfassen, und ein Buch kann von mehreren Autoren geschrieben sein. Anderes Beispiel: Die Firma des Sachbearbeiters wurde umstrukturiert, Teamwork ist angesagt. Er ist jetzt zusammen mit einigen seiner Kollegen für mehrere Bereiche zuständig, und jedem Sachbearbeiter können unterschiedliche Bereiche zugeordnet werden. Es entsteht ein komplexes Beziehungsgeflecht. Wenn man jetzt einen Bereich angibt, dann liefert die Beziehung nicht mehr diesen Sachbearbeiter allein, sondern das Team, das diesen Bereich bearbeitet.

Abb. 1–6
Eine m:n-Beziehung

Die Menge aller Beziehungen zwischen zwei Entitäten `E1` und `E2` ist das kartesische Produkt, das ist die Menge aller möglichen Paare `(E1,E2)`. Auch diese Paare können Attribute besitzen, in denen die Eigenschaften der Beziehung näher beschrieben sind. Diese Attribute können keiner der beteiligten Entitäten zugeordnet werden, da sie zu beiden in

einer Beziehung stehen, sie beschreiben die Eigenschaften der Paare. Ihnen werden nur dann Werte zugewiesen, wenn die Beziehung zustande kommt. Wenn Sie in einem Onlinekatalog eine Bestellung abschicken, kommt die Beziehung zwischen Ihnen als Kunden und dem bestellten Artikel zustande. Werte, die dieser Beziehung zugeordnet werden, sind zum Beispiel das Bestelldatum und der Gesamtpreis der Bestellung.

1.4.4 Schlüsselattribute und Primärschlüssel

Instanzen von Entitäten müssen der Forderung genügen, eindeutig identifizierbar zu sein. PostgreSQL überwacht diese Forderung und lässt normalerweise nicht zu, dass Sie zwei identische Datensätze in eine Tabelle einfügen. (Seit der Version 7.2 kann diese Standardeinstellung aufgehoben werden.) Jede Ausprägung einer Entität muss mindestens ein Attribut oder eine Kombination von Attributen besitzen, die sie von allen anderen Datensätzen in der Tabelle eindeutig unterscheidet.

Schlüsselkandidaten

Solche Attribute oder Attributmengen heißen Schlüsselkandidaten. Sie sind alle geeignet, einen Datensatz zu identifizieren. Wenn ein Schlüsselkandidat mehrere Attribute umfasst, spricht man von einem zusammengesetzten Schlüssel, und die einzelnen Attribute darin werden als Schlüsselattribute für diesen Entitätstyp bezeichnet. Ein Entitätstyp kann mehrere Schlüsselkandidaten haben. Beim Entwurf wird einer davon als Identifikationskriterium bestimmt. Diesen nennt man Primärschlüssel.

Primärschlüssel

Welchen Schlüsselkandidaten man als Primärschlüssel auszeichnet, ist unerheblich. Entscheidend ist nur, dass dieser Primärschlüssel die Datensätze eindeutig identifiziert. Bei zusammengesetzten Schlüsseln sollte man darauf achten, mit möglichst wenig Attributen auszukommen. Daraus folgt, dass immer ein Schlüssel existiert, da ein trivialer Schlüssel, der alle Attribute umfasst, den Datensatz immer identifizieren kann. Für Primärschlüssel gilt wegen der Eindeutigkeit auch, dass der Wert dieses Attributs niemals undefiniert sein darf. Kann man aus den Eigenschaften eines Entitätstyps keinen eindeutigen Schlüssel bilden außer dem trivialen Schlüssel, der alle Attribute umfasst, wird ein zusätzliches, künstliches Attribut eingefügt, um zu gewährleisten, dass jede Ausprägung eindeutig ist. Solche zusätzlichen Attribute oder Unique Identifiers (UID) sind wohl bekannt: als Personalnummern, Artikelnummern, Kundennummern...

1.4.5 Schwache Entitäten

Im ER-Modell wird eine Entität schwach genannt, wenn ihre Existenz von der Existenz einer Hauptentität abhängig ist und sie selber keinen Primärschlüssel hat. Dazu ein Beispiel: Zur Verwaltung des Onlineshops gebe es eine Tabelle mit den Daten zu den verschiedenen Lieferanten und dem Primärschlüssel L-ID. Mit einigen der Lieferanten wurden Zusatzvereinbarungen getroffen. Diese internen Vermerke können nun in einer separaten Tabelle gespeichert werden, die als Primärschlüssel die L-ID der Lieferantentabelle besitzt. Wenn ein Lieferant nicht in der Haupttabelle vorhanden ist, hat er keine L-ID und somit kann es auch keinen internen Vermerk zu diesem Lieferanten geben.

1.5 Von der realen Welt zum Modell

Der Entwurf einer Datenbankstruktur ist eine der wichtigsten Aufgaben. Je nach Projektgröße liegt ein Pflichtenheft oder eine Beschreibung der Funktionalität vor, die die künftige Anwendung erfüllen soll. Aus der Analyse dieser Beschreibungen müssen die Entitäten, ihre Attribute, ihre Beziehungen und eventuell einzuhaltende Integritätsbedingungen abgeleitet werden. Die wichtigsten Forderungen an dieses Datenmodell sind die Widerspruchsfreiheit und die Vermeidung von Redundanz.

Lassen Sie uns eine Beispielanwendung skizzieren: Auf einer Internetseite sollen Publikationen verschiedener Autoren veröffentlicht werden. Sie haben die Aufgabe, Webseiten zu erstellen, auf denen

- die Autoren nach Namen sortiert ausgegeben werden,
- die Publikationen eines Autors aufgelistet werden oder
- die Publikationen entweder nach Titel, Themenbereich oder Erscheinungsdatum sortiert angezeigt und ausgegeben werden sollen.

1.5.1 Schritt 1: Entitäten identifizieren

Aus dieser kurzen Beschreibung kann man ohne großes Nachdenken zwei Entitäten bestimmen: `Autor` und `Publikation`. Die Entität `Autor` muss mindestens das Attribut `Name` haben. Da Attributwerte unteilbar (atomar) sein müssen, wird das Attribut `Name` in `Vorname` und `Nachname` aufgeteilt. Die Entität `Publikation` muss mindestens die Attribute Thema, Titel und Erscheinungsdatum enthalten. Im »richtigen Leben«

wird man zu den beiden Entitäten weitere Daten speichern wollen und dementsprechend weitere Attribute definieren.

Wenn ein Pflichtenheft oder eine Beschreibung vorliegt, in der die Aufgaben der zukünftigen Applikation beschrieben sind, kann man meistens aus wiederholt vorkommenden Substantiven Entitäten ableiten. In der obigen Aufgabenbeschreibung treten die Substantive »Autor« und »Publikation« wiederholt auf. Die Attribute müssen so gewählt werden, dass sie die gestellte Aufgabe erfüllen. Wenn die Publikationen nach ihrem Erscheinungsdatum sortiert angezeigt werden sollen, so muss das Modell ein Attribut für dieses Datum enthalten. Alle Daten, die gespeichert werden sollen, müssen als Attribut modelliert und den Entitäten zugeordnet werden.

Ein anderer Weg ist, zuerst alle Daten, die gespeichert werden sollen, aufzuschreiben und dann als Attribute den Entitäten zuzuordnen. Wenn dann aus der Liste nicht alle Attribute zugeordnet werden konnten, fehlt Ihnen möglicherweise eine Entität in Ihrem Entwurf.

1.5.2 Schritt 2: Beziehungen finden

Das Auffinden von Beziehungen ist nicht immer einfach. Hilfreich ist eine Tabelle, bei der links und oben die Entitäten angeschrieben werden und dann versucht wird, auf die Schnittpunkte ein Verb zu schreiben, das die Beziehung der beiden Entitäten charakterisiert. Jede Spalte und jede Zeile der Matrix sollte mindestens einen Eintrag haben, ansonsten haben Sie eine Entität, die nicht mit dem Rest des Systems interagiert. Überprüfen Sie, ob es Sinn macht, diese Entität beizubehalten, oder ob Sie diese Entität als Attribut einer anderen Entität zuschlagen können.

In dem Beispiel stehen die beiden Entitäten `Autor` und `Publikation` in der natürlichen Beziehung zu `schreibt`.

1.5.3 Schritt 3: Kardinalität der Beziehungen festlegen

Welche Zuordnungen gibt es zwischen den verschiedenen Entitäten und welche Kardinalität haben sie? Um das Beispiel aufzugreifen: Handelt es sich hier um Publikationen, die jeweils von genau einem Autor stammen, wie etwa Schüleraufsätze, oder können die Publikationen von einem Autorenteam verfasst worden sein? Im ersten Fall wird eine `1:n`-Beziehung modelliert, in der ein Autor zwar mehrere Publikationen verfassen kann, die Publikationen aber alle nur von einem Autor allein stammen. Lässt man mehrere Autoren als Verfasser einer Publikation zu, entsteht eine `m:n`-Beziehung, in der Autoren,

allein oder im Team, mehrere Publikationen verfassen können. Solche semantischen Fragen müssen im konzeptuellen Schema festgelegt werden. Es sind wichtige Entwurfsentscheidungen, die die Datendefinition bestimmen und deren spätere Änderung sehr aufwändig werden kann.

1.5.4 Schritt 4: Entitätstypen auf Eindeutigkeit prüfen

Künstliches Attribut als Schlüsselattribut

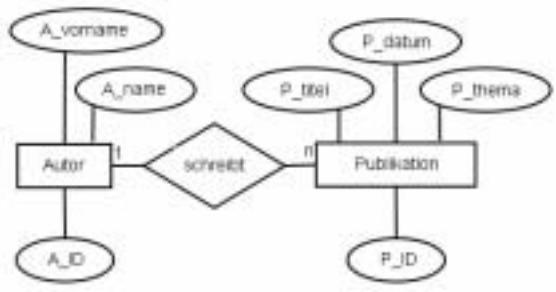
In der Praxis wird es wohl kaum vorkommen, dass am selben Tag zwei Publikationen zu demselben Thema mit demselben Titel herauskommen. Man kann davon ausgehen, dass diese Attribute den Entitätstyp eindeutig bestimmen. Ganz anders sieht es bei den Autoren aus: Namensgleichheiten, selbst bei Vor- und Nachnamen, sind keine Seltenheit, so dass wir hier ein künstliches Attribut als Schlüsselattribut einfügen, die `A_ID`.

Auch wenn bei den Publikationen nicht mit zwei identischen Datensätzen zu rechnen ist, ist es in der Praxis nicht sinnvoll, jeden Datensatz über die Kombination aller seiner Attributwerte anzusprechen. Auch in diesem Fall wird man einen eindeutigen Identifikator (`P_ID`) vergeben. Die Attribute haben jeweils den Anfangsbuchstaben der entsprechenden Entität als Präfix. Das ist keine Vorschrift, es erleichtert die Zuordnung eines Attributs zu einer Tabelle.

1.5.5 Schritt 5: Erstellen des Entity-Relationship-Modells

Gehen wir zunächst davon aus, dass ein Autor zwar mehrere Publikationen verfassen kann, eine Publikation aber immer nur einen Autor hat. Dadurch wird die Kardinalität der Beziehung auf `1:n` festgelegt. Aus diesen Vorgaben lässt sich das folgende Entity-Relationship-Modell ableiten:

Abb. 1–7
Autor – schreibt – Publikation



1.5.6 Schritt 6: Das Modell an der Wirklichkeit prüfen

Nachdem Sie das Modell entworfen haben, überprüfen Sie, ob Sie die geplante Anwendung anhand dieses Modells nachvollziehen können. Nehmen Sie die Aufgabenbeschreibung und prüfen Sie, ob Ihr Datenmodell logisch vollständig ist, das heißt, ob Sie für jede Aufgabe alle benötigten Attributwerte über Beziehungen erreichen können. Um alle Publikationen eines Autors auf einer Webseite anzeigen zu können, muss eine Beziehung zwischen den Entitäten `Autor` und `Publikation` in dem Modell existieren.

Entitäten:

- Hat jede Entität einen Primärschlüssel?
- Hat jede Entität mindestens zwei Attribute? Wenn nein, dann wäre dieses einzige Attribut der Primärschlüssel, der nichts referenzieren würde. Damit wäre die Entität nutzlos.
- Beschreiben die Attribute den Entitätstyp ausreichend oder sind Attribute im Modell enthalten, die für die Anwendung nicht gebraucht werden?
- Gibt es mindestens eine Beziehung zu einer anderen Entität? Wenn nein, dann haben Sie entweder eine Beziehung in Ihrem Entwurf vergessen oder die Entität wird nicht gebraucht. Es sei denn, Sie stellen noch andere Daten zur Verfügung, die nicht direkt mit der Anwendung verknüpft sind. Zum Beispiel könnten Sie als Service eine Liste mit Links zu ähnlichen Internetseiten ausgeben.

Attribute:

- Gibt es für jedes Attribut in einer Instanz nur einen Wert, der nicht weiter zerlegbar ist? Wenn es ein Attribut gibt, dessen Wert aus mehreren Werten zusammengesetzt ist, dann fehlt der Entität ein Attribut, oder es fehlt eine Entität in dem Modell. Ein solcher Attributwert ist der Name, der in Vorname und Nachname aufgeteilt werden kann.

Beziehungen:

- Entsprechen die Kardinalitäten der Beziehungen dem tatsächlichen Realweltausschnitt und der Beschreibung im konzeptuellen Schema?
- Gibt es Beziehungen in dem Modell, die bereits durch andere Beziehungen ausgedrückt wurden und damit überflüssig sind? Zum Beispiel: Eine Publikation besteht aus Kapiteln und diese wiederum aus Unterkapiteln. Dann ist es logisch, dass eine Publikation auch

aus Unterkapiteln besteht; die Modellierung dieser Beziehung ist aber unnötig.

1.6 Vom Modell zum Datenbankentwurf

Es gibt kein allgemein gültiges Entwurfsverfahren.

Im nächsten Schritt gilt es nun, das erstellte Entity-Relationship-Modell in einen Datenbankentwurf zu überführen, in dem die Informationen möglichst redundanzfrei gespeichert und möglichst einfach und korrekt abgefragt werden können. Um es vorweg zu sagen: Es gibt kein allgemein gültiges Entwurfsverfahren. Die Modellierung des Realweltausschnitts kann zu unterschiedlichen ER-Modellen und damit zu unterschiedlichen Datenbankentwürfen führen. Es gilt nun, unter allen diesen Alternativen einen günstigen Entwurf zu finden, in dem keine Informationen, vor allem keine Beziehungen, verloren gehen.

1.6.1 Die Umsetzung von Entitätstypen

Ein Entitätstyp wird als Relation (Tabelle) dargestellt. Alle Attribute des Entitätstyps werden zu Attributen der Relation, und der Primärschlüssel des Entitätstyps wird zum Primärschlüssel der Relation.

Der Entitätstyp `Publikation` aus dem obigen Beispiel wird als Relation definiert. Mit dem Präfix `P_` wird die Zuordnung eines Attributs zu der Entität `Publikation` gekennzeichnet. Dies ist keine Vorschrift, erleichtert aber die Lesbarkeit:

```
Publikation ( P_ID, P_thema, P_titel, P_datum )
```

Die entsprechende Tabelle sieht folgendermaßen aus:

Abb. 1–8
Die Tabelle `Publikation`

P_ID	P_thema	P_titel	P_datum
3	UML	UML – Möglichkeiten und Grenzen	30.06.2000
17	VisualBasic	Grafische Oberflächen mit VB	14.01.2001
36	PHP	Dateiuploads leichtgemacht	23.09.2001

1.6.2 Die Umsetzung von Beziehungstypen

1:1-Beziehungen

Nehmen wir an, dass es eine Beziehung zwischen zwei Entitätstypen `E1` und `E2` gibt, so dass jede Ausprägung von `E1` genau zu einer Ausprägung von `E2` in Beziehung steht. (Etwa `E1` = Mann, `E2` = Frau und die Beziehung = `ist verheiratet mit`.) In solchen Fällen kann man die

Daten in einer einzigen Tabelle verwalten, die alle Attributwerte beider beteiligter Entitäten enthält. (VornameMann, NachnameMann, …, VornameFrau, NachnameFrau, …)

Fremdschlüssel

Wenn es zu jeder Ausprägung von E1 höchstens eine Ausprägung von E2 gibt oder auch keine, ist es in den meisten Fällen sinnvoll, jede Entität in einer Tabelle zu modellieren. Wenn im obigen Beispiel auch unverheiratete Personen in die Tabelle aufgenommen werden, bleiben die Felder für den Ehepartner leer.

Oder manche Bücher des Onlineshops liegen auch in einer englischen Version vor. Dann besteht zwischen der Entität Buch und der neuen Entität engl_Version eine 1:1-Beziehung, die aber nur einen Teil aller Bücher betrifft. Wenn man diese neuen Attribute in die Buchtabelle übernimmt, bleiben sehr viele Tabellenzellen leer, weil es für viele Titel keine englische Version gibt. Die Beziehung wird dadurch modelliert, dass man der neuen Tabelle engl_Version zu den eigenen Attributen eine Spalte für den Primärschlüssel der Entität Buch zufügt. Sollte die Beziehung selber Attribute besitzen, werden diese ebenfalls als zusätzliche Spalten in die neue Tabelle übernommen. Wenn ein Primärschlüssel als zusätzliches Attribut in eine fremde Tabelle übernommen wird, nennt man ihn Fremdschlüssel.

1:n-Beziehungen

Angenommen, die Publikationen sollen kapitelweise auf der Webseite ausgegeben werden, nachdem der Besucher ein Kapitel ausgewählt hat. Als Folge muss die Entität Publikation genauer spezifiziert werden. Ein Attribut Kapitel zuzufügen ist nicht möglich, da dies mit Sicherheit kein atomarer Wert ist. Dann bleibt die Möglichkeit, eine neue Entität Kapitel in das Modell einzufügen und ihr die notwendigen Attribute wie Seitenzahl, Überschrift sowie eine eindeutige Kapitelnummer mitzugeben. Diese neue Entität steht mit Publikation in der Beziehung ist Teil von und die Kardinalität der Beziehung ist n:1 oder 1:n, je nach Blickrichtung (ein Kapitel gehört zu genau einer Publikation, die Publikation hat n Kapitel).

Oder eine Publikation werde von genau einem Autor verfasst und ein Autor kann mehrere Publikationen schreiben. Dann steht die Entität Autor mit der Entität Publikation in der Beziehung schreibt und die Kardinalität ist 1:n. Diesem Beispiel entspricht das ER-Modell in Abbildung 1–7 weiter oben.

1:n-Beziehungen werden so aufgelöst, dass die Entität, die am n-Ende einer Beziehung liegt, den Primärschlüssel der anderen beteiligten Entität als Fremdschlüssel übernimmt und so den zugehörigen Daten-

satz in dieser Entität referenzieren kann. Abbildung 1–9 zeigt eine aufgelöste 1:n-Beziehung, in der die Tabelle Publikation den Primärschlüssel A_ID aus der Tabelle Autor als Fremdschlüssel übernommen hat. Dadurch ergeben sich die beiden Relationen:

```
Autor (A_ID, A_vorname, A_name)
Publikation (P_ID, A_ID, P_thema, P_titel, P_datum)
```

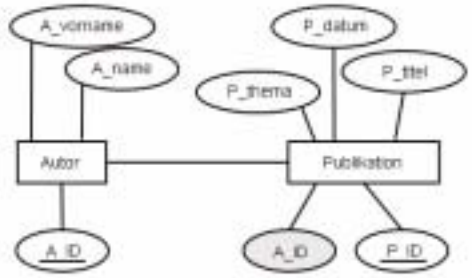


Abb. 1–9
Auflösung einer 1:n-Beziehung

Dieses abstrakte Modell auf Tabellen übertragen zeigt Abbildung 1–10. Der in die Tabelle Publikation übernommene Fremdschlüssel ist grau eingefärbt.

Abb. 1–10
Tabellen in einer 1:n-Beziehung

A_ID	A_vorname	A_name
11	Bruno	Maier
7	Josef	Amann
27	Lisa	Amann
16	Hugo	Müller

P_ID	A_ID	P_thema	P_titel	P_datum
3	11	UML	UML - Möglichkeiten und Grenzen	30.06.2000
17	7	VisualBasic	Grafische Oberflächen mit VB	14.01.2001
36	11	PHP	Dateiuploads leichtgemacht	23.09.2001
...	...	...	...	...

m:n-Beziehungen

Um m:n-Beziehungen aufzulösen, wird eine neue Entität in das Modell eingefügt, die die Primärschlüssel der beiden beteiligten Entitäten als Fremdschlüssel enthält und zusätzlich noch etwaige Attribute der Beziehung. Dadurch ergeben sich zwischen jeder alten Entität und der

neuen Entität jeweils 1:n-Beziehungen (die n-Seite liegt bei der neuen Entität).

Entsprechend der Aufgabenstellung unseres Beispiels sollen Publikationen entweder nach dem Titel, dem Erscheinungsdatum oder dem Thema sortiert angezeigt werden. In der Praxis werden Publikationen oft nach mehreren Themengebieten katalogisiert, was in unserem bisherigen Modell nicht möglich ist. Die Entität Publikation enthält ein Attribut P_thema, mit dem nur eine Zuordnung zu einem Themengebiet möglich ist, da mehrwertige Attributwerte im relationalen Modell nicht erlaubt sind. Somit muss man eine neue Entität Katalog in das Modell einfügen und ihr beispielsweise die Attribute T_bereich und T_inhalt, für den Themenbereich und eine inhaltliche Kurzbeschreibung dieses Bereichs, zuordnen. Dadurch ergibt sich zwischen den Entitäten Publikation und Katalog eine n:m-Beziehung, denn eine Publikation kann mehreren Themenbereichen zugeordnet werden und zu einem Themenbereich kann der Katalog viele Publikationen enthalten. (Das bisherige Attribut P_thema entfällt.) Das führt zu den Relationen

```
Autor (A_ID, A_name, A_vorname)
Publikation (P_ID, P_titel, P_datum)
Katalog(T_bereich, T_inhalt)
```

und zu dem folgenden ER-Modell:

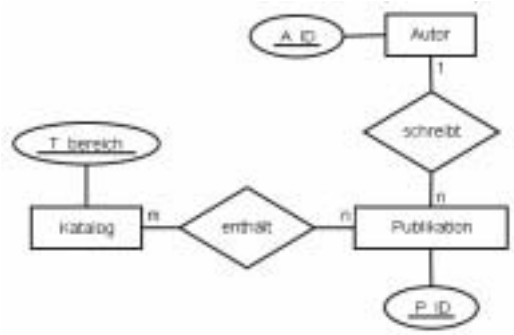


Abb. 1–11
Das ER-Modell der drei Entitäten

Nach der Auflösung der m:n-Beziehung zwischen Katalog und Publikation erhalten wir eine neue Relation Pub_Katalog, die Primärschlüssel der beiden beteiligten Relationen als Fremdschlüssel enthält. Diese Relation ist in unserem Beispiel eine reine Verknüpfungstabelle, sie kann aber auch etwaige Attribute der Beziehung enthalten.

```
Pub_Katalog (P_ID, T_bereich)
```

Abb. 1–12
Die Umsetzung einer m:n-Beziehung

Die Übertragung dieses Modells auf richtige Tabellen sehen Sie in Abbildung 1–13. Diese Art der Darstellung wird mit steigendem Füllungsgrad der Tabellen sehr schnell unübersichtlich. Es sollte nur an einem konkreten Beispiel veranschaulicht werden, wie eine `m:n`-Beziehung aufgelöst wird und wie Datensätze durch die Übernahme von Fremdschlüsseln miteinander verknüpft sind. (Die Fremdschlüssel sind grau unterlegt.)

In den letzten Abschnitten wurde gezeigt, wie Sie aufgrund einer Aufgabenstellung ein logisches Datenmodell entwickeln und dieses Modell in eine Datenbankstruktur überführen. Gleichzeitig ist der letzte Abschnitt ein Beispiel dafür, wie ein bestehendes Modell erweitert werden kann, wenn sich die Semantik der Anwendung ändert. Aus dem Attribut `P_thema` der Entität `Publikation` wurde eine neue Entität `Katalog` modelliert und in das Modell eingefügt. Über das Attribut `T_bereich` ist die neue Entität mit der Entität `Publikation` verknüpft.

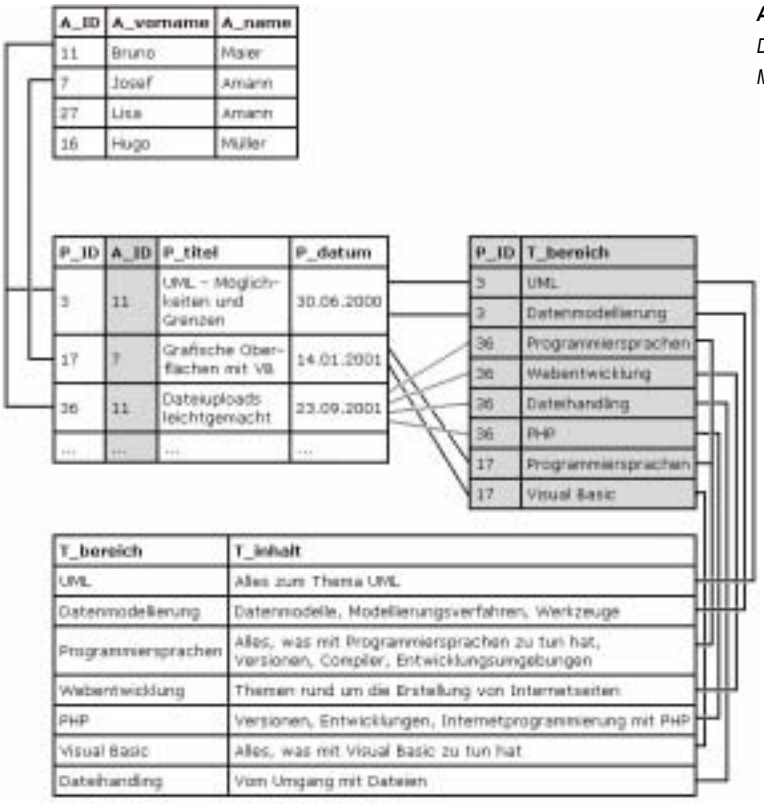
Die wichtigste Voraussetzung für eine übersichtliche und änderungsfreundliche Datenbankstruktur ist eine korrekte Analyse der Aufgabenstellung und der zu modellierenden Daten.

1.7 Das relationale Datenmodell

Eine Darstellung der relationalen Datenbanken wäre ohne Bezug auf die mathematische Basis, die dem Relationenmodell zugrunde liegt, unvollständig.

Mathematische Grundlagen

Das relationale Datenmodell wurde 1970 von C. F. Codd vorgestellt. Die mathematischen Grundlagen dieses Modells sind die mengentheoretischen Relationen (nicht zu verwechseln mit den Relation-



A_ID	A_vorname	A_name
11	Bruno	Maier
7	Josef	Amann
27	Lisa	Amann
16	Hugo	Müller

P_ID	A_ID	P_titel	P_datum
3	11	UML – Möglich-keiten und Grenzen	30.06.2000
17	7	Grafische Ober-flächen mit VB	14.01.2001
36	11	Dateiuploads leichtgemacht	23.09.2001
...	...	...	...

P_ID	T_bereich
3	UML
3	Datenmodellierung
36	Programmiersprachen
36	Webentwicklung
36	Dateihandling
36	PHP
17	Programmiersprachen
17	Visual Basic

T_bereich	T_inhalt
UML	Alles zum Thema UML
Datenmodellierung	Datenmodelle, Modellierungsverfahren, Werkzeuge
Programmiersprachen	Alles, was mit Programmiersprachen zu tun hat, Versionen, Compiler, Entwicklungsumgebungen
Webentwicklung	Themen rund um die Erstellung von Internetseiten
PHP	Versionen, Entwicklungen, Internetprogrammierung mit PHP
Visual Basic	Alles, was mit Visual Basic zu tun hat
Dateihandling	Vom Umgang mit Dateien

Abb. 1–13
Die Übertragung des Modells auf Tabellen

ships im ER-Diagramm). Sie sind als Teilmengen des kartesischen Produkts zweier oder mehrerer Mengen definiert.

`D1` sei die Menge `{Anne, August, Achim}`, `D2` sei die Menge `{Ulm, Kiel, Bonn}`, dann ist das kartesische Produkt die Menge aller möglichen Paare aus diesen beiden Mengen.

Anne	Ulm
Anne	Kiel
Anne	Bonn
August	Ulm
August	Kiel
August	Bonn
Achim	Ulm
Achim	Kiel
Achim	Bonn

Abb. 1–14
Das kartesische Produkt Tupel

Jede Teilmenge dieser Menge ist eine zweistellige Relation über den Wertemengen D1 x D2, die nach einer bestimmten Zuordnungsvorschrift gebildet wurde. Diese Relation kann auch leer sein oder alle Paare umfassen und ist zweistellig, weil die Relation über zwei Mengen gebildet wurde. Ein einzelnes Element aus einer Relation nennt man in der Mathematik Tupel. Dieser Begriff wurde auch in die Datenbankterminologie übernommen, wo ein Tupel einen Datensatz repräsentiert.

Einen Entitätstyp kann man sich als zweidimensionale Tabelle vorstellen, in der jede Ausprägung E1, E2 ... eine Zeile repräsentiert und jedes Attribut (A1, A2, ... ,An) eine Spalte. Den Attributen A1...An liegen jeweils die Wertebereiche (Domains) D1...Dn zugrunde.

Abb. 1–15
Ein Entitätstyp mit zwei Attributen

R	A1	A2
E1	Anne	Ulm
E2	August	Kiel
E3	Achim	Bonn

Jede Tabelle oder Teile davon können als n-stellige Relationen über diesen Wertebereichen aufgefasst werden. R ist eine Menge von geordneten Tupeln über dem Wertebereich D1 x D2 ... Dn und damit eine Teilmenge des kartesischen Produkts. Eine Zeile der Tabelle entspricht einem Tupel r = (a1, a2 ... an) mit r Element von R und a1 Element von D1, a2 Element von D2 usw. In Abbildung 1–15 ist R die Menge der Tupel E1(Anne, Ulm), E2(August, Kiel) und E3(Achim, Bonn) und wie Sie anhand Abbildung 1–14 leicht feststellen können, ist R eine Teilmenge des kartesischen Produkts. Da jeder Attributwert aus einem definierten Wertebereich stammen muss, kann es im relationalen Datenmodell nicht vorkommen, dass ein Attribut keinen Wert hat. (Wenn man als Wertebereich eines Attributs die leere Menge zugrunde legen würde, wäre dieses Attribut in jedem Tupel leer und damit sinnlos.) Jedes Datenbanksystem stellt Ihnen deshalb einen so genannten Nullwert zur Verfügung, der die semantische Bedeutung »undefiniert« oder »nicht bekannt« hat.

Relationen als Mengen von Tupeln

Da Relationen als Mengen von Tupeln definiert sind und in Mengen keine identischen Elemente auftreten können, folgt aus dieser Definition bereits, dass die Tupel in einer Relation eindeutig sein müssen. Übertragen auf eine Tabelle bedeutet das, die einzelnen Zeilen müssen eindeutig sein. Die beiden Begriffe, Tabelle und Relation, werden in der Datenbankwelt synonym benutzt.

1.8 Normalisierung

Weiter oben wurde bereits ausgeführt, dass eine Tabelle genau einen Entitätstyp mit allen seinen Attributen beschreiben sollte. Dadurch werden die Informationen, die logisch zusammengehören, auch zusammen in einer Tabelle gespeichert. Diese allgemeine Forderung soll an einem Beispiel verdeutlicht werden. Wir betrachten eine Tabelle, in der alle Daten zu einer `Publikation` verwaltet werden, wobei eine `Publikation` nur einen `Autor` haben soll:

```
Publikation (P_ID, P_titel, P_datum, A_ID, A_vorname, A_name,
T_bereich, T_inhalt),
```

{`P_ID`, `T_bereich`} sei der Schlüssel.

Bei genauerem Hinsehen weist dieses Relationenschema Schwächen auf, die weitreichende Konsequenzen haben:

- Wenn ein Autor mehrere Publikationen geschrieben hat, werden seine Daten jedes Mal mit jeder Publikation gespeichert.
- Wenn eine Publikation mehreren Themenbereichen zugeordnet wird, werden die Publikationsdaten und die Autorendaten jedes Mal mitgespeichert.

Man erhält eine Tabelle, in der sehr viele Daten mehrfach gespeichert sind. Redundanzfreiheit als Entwurfsziel ist damit nicht zu erreichen. Außerdem treten hier so genannte Anomalien auf:

- Einfüge-Anomalie (Insertion Anomaly)

Es kann keine neue Publikation aufgenommen werden, wenn sie keinem bestimmten Themenbereich zugeordnet werden kann. Oder es können nur Autoren, die mindestens eine Publikation geschrieben haben, in die Tabelle eingefügt werden. Der Grund dafür ist, dass in einem Schlüsselattribut keine Nullwerte erlaubt sind.

- Lösch-Anomalie (Deletion anomaly)

Wird ein Themenbereich aus dem Katalog gelöscht, so werden alle Publikationen, die diesem Bereich zugeordnet waren, mit aus der Tabelle gelöscht. Wird eine Publikation gelöscht, verschwinden damit auch die Daten des Autors sowie der Themenbereich.

- Änderungs-Anomalie (Update Anomaly)

Ändert sich der Name eines Autors, so muss die ganze Tabelle durchsucht und jede Zeile, die den Autor enthält, geändert werden. Obwohl sich nur ein einzelner Attributwert ändert, müssen in der Tabelle mehr-

fache Änderungen durchgeführt werden, weil die Daten redundant gespeichert wurden.

Die Redundanz und alle diese Anomalien sind darin begründet, dass in der Tabelle `Publikation` Informationen über mehrere Entitätstypen gespeichert sind: Über Publikationen, Autoren und den Themenkatalog, obwohl der Name eines Autors nichts mit dem Themenbereich zu tun hat, der einer Publikation zugeordnet wurde.

Normalisierung als Instrument zur Qualitätssicherung

Um eine widerspruchsfreie Datenverwaltung zu gewährleisten, in der die oben genannten Anomalien nicht mehr auftreten, wurden eine Reihe von Entwurfsregeln definiert – die Normalisierung. Sie ist ein Instrument zur Qualitätssicherung und hat zum Ziel, ein redundanzfreies Modell zu entwickeln, in dem alle Informationen genau einmal gespeichert sind und in dem keine der oben genannten Anomalien auftritt. Ausgangsbasis und zentraler Begriff für die Normalisierung ist die funktionale Abhängigkeit.

1.8.1 Funktionale Abhängigkeit

Semantische Integritätsbedingungen

Funktionale Abhängigkeiten sind das wichtigste Konzept, mit dem die Beziehungen zwischen Attributen einer Relation charakterisiert werden. Aussagen über diese Beziehungen können nur von der Realwelt abgeleitet werden und müssen für alle möglichen Zeilen in einer Tabelle gültig sein. Sie stellen die semantischen Integritätsbedingungen dar. Das bedeutet, dass solche Aussagen nicht von einem aktuellen Wert eines Attributs abhängen dürfen. Um ein einfaches Beispiel zu geben: Jede Person hat genau ein Geburtsdatum, damit ist dieses Geburtsdatum funktional abhängig von der Person. Wäre dies nicht so, könnte man einer Person mehrere Geburtsdaten zuordnen.

Wenn in zwei Zeilen unserer Tabelle der Wert von `A_ID` gleich ist, so müssen immer auch die Werte von `A_vorname` und `A_name` dieselben sein. Diese Aussage wurde von der Realwelt abgeleitet, in der ein Autor nicht plötzlich einen anderen Vornamen haben kann. Daraus folgt, dass es zwischen `A_ID` und `A_vorname` bzw. zwischen `A_ID` und `A_name` eine funktionale Abhängigkeit gibt.

Etwas formaler: In einer Relation `R` ist das Attribut `B` genau dann von dem Attribut `A` funktional abhängig, wenn alle Tupel aus dieser Relation, die für `A` den Wert `X` haben, auch für `B` den Wert `Y` haben, wenn also das Paar `A, B` immer dieselben Wertepaare besitzt bzw. wenn der Wert von `A` den Wert von `B` bedingt. Man schreibt dafür `A->B`. Anders ausgedrückt: Für jeden Wert von `A` existiert genau ein Wert für `B`.

Volle funktionale Abhängigkeit

Ein Attribut wird als voll funktional abhängig von einer Menge von Attributen bezeichnet, wenn es von allen diesen Attributen funkti-

onal abhängig ist und nicht nur von einem Teil der Attribute. So kann ein Geburtsdatum nicht nur von einem Attribut Vorname abhängig sein, sondern muss von den Attributen, die diese Person eindeutig beschreiben, abhängen.

Untersuchen wir nun die funktionalen Abhängigkeiten, die in der obigen Relation gegeben sind. Zunächst muss ein Schlüssel gefunden werden, der geeignet ist, alle Datensätze eindeutig zu identifizieren. Die Attributmenge {P_id, T_bereich} erfüllt diese Forderung. Wenn der Identifikator der Publikation bekannt ist, kann man auf den Autor schließen, denn jeder Publikation ist genau ein Autor zugeordnet. Mit dem Attribut T_bereich kann man dann alle Datensätze dieser Publikation eindeutig bestimmen. Mehr Attribute sind dazu nicht erforderlich. {P_ID, T_bereich} ist eine minimale Attributmenge, die zur Identifikation der Datensätze ausreicht, und wird deshalb als Primärschlüssel ausgezeichnet.

```
Publikation (P_ID, P_titel, P_datum, A_ID, A_vorname, A_name,
T_bereich, T_inhalt) mit dem Primärschlüssel {P_ID, T_bereich}.
```

Dann erhalten wir die folgenden funktionalen Abhängigkeiten:

```
{P_ID, T_bereich} -> P_ID
{P_ID, T_bereich} -> T_bereich
```

Einzelne Schlüsselattribute sind von zusammengesetzten Schlüsselattributen, in denen sie selber enthalten sind, voll funktional abhängig, denn jedem Paar {P_ID, T_bereich} ist genau eine Publikation bzw. genau ein Themenbereich zugeordnet, nämlich genau die, die in dem Paar {P_ID, T_bereich} enthalten sind.

```
{P_ID} -> P_titel
```

jede Publikation hat genau einen Titel,

```
{P_ID} -> P_datum
```

jede Publikation hat genau ein Erscheinungsdatum,

```
{P_ID} -> A_ID
```

jeder Publikation ist genau ein Autor zugeordnet,

```
{T_bereich} -> T_inhalt
```

jeder Themenbereich hat genau eine Kurzbeschreibung,

```
A_ID -> A_name
```

jedem Autor ist genau ein Name zugeordnet, und damit auch:

```
{P_ID} -> A_name
```

jeder Publikation ist genau ein Autorenname zugeordnet,

```
A_ID -> A_vorname
```

jedem Autor ist genau ein Vorname zugeordnet, und damit auch:

```
{P_ID} -> A_vorname
```

jeder Publikation ist der Vorname des Autors zugeordnet.

Nachdem nun die funktionalen Abhängigkeiten der Relation ermittelt sind, gehen wir weiter zu den Normalisierungsregeln.

1.8.2 Die erste Normalform – 1NF

Die Definition der ersten Normalform ist recht einfach. Jede Relation, deren Attributwerte unteilbar sind, ist bereits in der ersten Normalform. Wenn in einer Spalte Vorname und Nachname einer Person enthalten sind, ist die Relation demnach nicht in der ersten Normalform. Die Forderung ist erfüllt, wenn in jeder Tabellenspalte nur ein einzelner Wert eingetragen ist. Die Relation

```
Publikation (P_ID, P_titel, P_datum, A_ID, A_name, A_vorname,
T_bereich, T_inhalt)
```

ist bereits in der ersten Normalform. Hier können sich allerdings Speicheranomalien ergeben, die die Konsistenz und Widerspruchsfreiheit des Systems gefährden. Einige Beispiele:

- Will man eine neue Publikation in die Relation einfügen, so ist das nur möglich, wenn der Autor bekannt ist und mit eingetragen wird (Einfüge-Anomalie),
- wenn man die Publikation in einen anderen Themenbereich einsortieren möchte, so muss man die Änderungen in allen Tupeln durchführen, in denen diese Publikation auftritt (Update-Anomalie),
- wenn man eine Publikation aus der Relation löscht, muss man das in allen Zeilen tun, in der die Publikation auftritt und löscht damit gleichzeitig den Autor der Publikation aus der Tabelle (Lösch-Anomalie).

1.8.3 Die zweite Normalform – 2NF

Die oben beschriebenen Anomalien lassen sich teilweise mit der Überführung der Relation in die zweite Normalform beheben. Hier kommen die funktionalen Abhängigkeiten zum Tragen. Eine Relation ist in der zweiten Normalform, wenn sie

- in der ersten Normalform ist und
- gleichzeitig jedes Nichtschlüsselattribut voll funktional abhängig vom Primärschlüssel ist.

Das bedeutet, dass es in der zweiten Normalform nur Relationen geben darf, in denen alle Attribute voll funktional vom Primärschlüssel abhängen. Wenn man die Menge der funktionalen Abhängigkeiten aus dem vorigen Abschnitt anschaut, stellt man fest, dass es außer den beiden Schlüsselattributen selbst kein Attribut in der Relation gibt, das voll funktional abhängig vom Primärschlüssel {P_ID, T_bereich} ist. Damit ist die Relation nicht in der zweiten Normalform, denn es gibt Attribute, wie etwa P_titel oder P_datum, die nur von einem Teil des Primärschlüssels , von P_ID, voll funktional abhängig sind.

Zerlegung der Relation

Um die Relation in die zweite Normalform zu überführen, muss sie zerlegt werden, und die Attribute müssen auf unterschiedliche Relationen verteilt werden. Alle Attribute, die nicht voll funktional abhängig vom Primärschlüssel sind, werden zusammen mit dem Teilschlüssel, von dem sie abhängen, in einer eigenen Relation zusammengefasst, so dass sich die folgenden Relationen ergeben:

```
Publikation (P_ID, P_titel, P_datum, A_ID, A_vorname, A_name)
Tkatalog (T_bereich, T_inhalt)
Pub_katalog (P_ID, T_bereich)
```

Die Relationen sind in der zweiten Normalform, da alle Nichtschlüsselattribute voll funktional von den Schlüsselattributen abhängig sind. Damit ist die Änderungs-Anomalie, die bei der ersten Normalform beschrieben ist, beseitigt, Publikationen können in verschiedene Themenbereiche eingetragen werden. Allerdings bestehen die Einfüge-Anomalie und die Lösch-Anomalie für die Relation Publikation immer noch.

1.8.4 Die dritte Normalform – 3NF

Die dritte Normalform beseitigt die beiden verbleibenden Anomalien. Schauen Sie die Relation Publikation und ihre funktionalen Abhängigkeiten an:

```
Publikation (P_ID, P_titel, P_datum, A_ID, A_vorname, A_name)
```

und die funktionalen Abhängigkeiten

```
{P_ID} -> A_ID
A_ID -> A_vorname
A_ID -> A_name
```

Damit gilt auch:

```
{P_ID} -> A_ID -> A_vorname
{P_ID} -> A_ID -> A_name
```

Transitive Abhängigkeiten

Solche Abhängigkeiten, über die man von Attribut zu Attribut »Ketten« bilden kann, heißen transitiv. A_vorname ist transitiv abhängig von dem Primärschlüssel P_ID. In der dritten Normalform sind transitive Abhängigkeiten nicht erlaubt. Sie sind verantwortlich für Anomalien wie die oben beschriebenen Einfüge- oder Löschanomalien. So ist es immer noch nicht möglich, einen Autor in die Tabelle einzutragen, der noch keine Publikation veröffentlicht hat, denn der Primärschlüssel der Relation ist P_ID und dieser kann nicht leer sein. Die dritte Normalform verbietet transitive Abhängigkeiten und beseitigt damit diese Anomalien.

Eine Relation R ist in der dritten Normalform, wenn sie

- in der zweiten Normalform ist und
- wenn kein Nichtschlüsselattribut transitiv abhängig vom Primärschlüssel ist. Es darf kein Nichtschlüsselattribut von einem anderen Nichtschlüsselattribut abhängig sein.
- Eine Relation in der zweiten Normalform mit nur einem Nichtschlüsselattribut ist automatisch in der dritten Normalform, denn bei nur zwei Attributen kann es keine transitiven Abhängigkeiten geben.

Die Relation Publikation ist nicht in der dritten Normalform, denn es bestehen zwei transitive Abhängigkeiten, wie oben gezeigt wurde. Wir zerlegen die Relation Publikation in zwei Relationen, so dass die transitive Abhängigkeit aufgebrochen wird:

```
Autor (A_ID, A_vorname, A_name)
Publikation (P_ID, P_titel, P_datum, A_ID)
```

So sind beide Relationen in der dritten Normalform. Wir erhalten eine separate Tabelle Autor und können Autoren einfügen, ohne dass zugleich eine Publikation eingetragen werden muss. Wenn eine Publikation gelöscht wird, werden nur ihre Daten gelöscht und nicht gleichzeitig die Daten des Verfassers. Und wenn sich der Name eines Autors ändert, muss diese Änderung nur an einer Stelle in der Tabelle Autoren durchgeführt werden. An diesen Beispielen sehen Sie, wie ein guter Entwurf die Konsistenzhaltung der Daten unterstützt.

Die ursprüngliche Relation wurde während des Normalisierungsprozesses in die folgenden Relationen zerlegt:

```
Autor (A_ID, A_name, A_vorname)
Publikation (P_ID, P_titel, P_datum, A_ID)
Pub_Katalog (P_ID, T_bereich)
TKatalog (T_bereich, T_inhalt)
```

Mit einem ER-Modell, in dem Entitäten auf Relationen abgebildet und die Umsetzungsregeln darauf angewendet werden, kommt man zu demselben Ergebnis (siehe Abbildung 1–12). In der Praxis ist mit dem Erreichen der dritten Normalform der Normalisierungsprozess oft abgeschlossen, obwohl immer noch Anomalien auftreten können, wie in dem folgenden Beispiel gezeigt wird.

1.8.5 Die Boyce-Codd-Normalform (BCNF)

Diese Normalform berücksichtigt auch Beziehungen zwischen Schlüsselattributen und ist damit restriktiver als die dritte Normalform: Funktionale Abhängigkeiten zwischen Nichtschlüsseln und Schlüsselattributen sind nicht erlaubt. Für jede funktionale Abhängigkeit `A -> B` muss gelten, dass `B` nicht in `A` enthalten sein darf und dass `A` ein Schlüsselattribut ist. Eine Relation ist in Boyce-Codd-Normalform, wenn kein Attribut funktional abhängig von einer Attributgruppe ohne Schlüsseleigenschaft ist.

Das Beispiel einer Relation `Mitarbeiter`:

```
Mitarbeiter (Personalnr, Abteilung, Name, Kuerzel)
```

Hier wird jedem Mitarbeiter zusätzlich zur Personalnummer ein Namenskürzel zugeordnet. Dieses Attribut ist eindeutig für jede Personalnummer. Dann sind {`Personalnr, Abteilung`} und {`Kuerzel, Abteilung`} jeweils zusammengesetzte Schlüsselkandidaten. Die Relation ist in 1NF, da die Attributwerte atomar sind, sie ist auch in der 2NF, da das einzige Nichtschlüsselattribut `Name` voll funktional abhängig von beiden Schlüsselkandidaten ist, und sie ist auch in der 3NF, da es außer `Name` keine weiteren Nichtschlüsselattribute gibt. Es gibt aber eine funktionale Abhängigkeit von Personalnummer und Kürzel, die nichts mit dem Schlüsselattribut `Abteilung` zu tun hat. In dieser Relation besteht immer noch eine Einfüge-Anomalie, da kein Mitarbeiter ohne Kürzel eingetragen werden kann.

Die Relation ist nicht in BCNF. `Personalnr` ist voll funktional abhängig von `Kuerzel`, `Kuerzel` ist aber nicht Schlüsselkandidat, sondern nur Teil eines Schlüsselkandidaten. Um die Relation in die BCNF zu überführen, wird diese funktionale Abhängigkeit in eine neue Tabelle ausgelagert. In der entstehenden Relation `Mitarbeiter` ist es

jetzt möglich, auch Mitarbeiter einzutragen, denen kein Namenskürzel zugeordnet wurde:

```
Mitarbeiter (Personalnr, Abteilung, Name)
Pnr_Kuerzel (Personalnr, Kuerzel)
```

Jede Relation in BCNF ist auch in der dritten Normalform. Die beiden Normalformen unterscheiden sich nur dann, wenn es in der Relation Schlüsselkandidaten mit überlappenden Attributen gibt.

Über die gezeigten Normalformen hinaus, die sich mit der Entfernung funktionaler Abhängigkeiten befassen, gibt es noch eine vierte und eine fünfte Normalform. Sie befassen sich mit mehrwertigen Abhängigkeiten und so genannten Join Dependencies. Sie spielen aber in der Praxis keine so große Rolle und werden deshalb hier nicht beschrieben.

1.8.6 Sinn und Unsinn der Normalisierung

Normalformen sind ein Mittel zum Zweck. Sie helfen, Anomalien zu vermeiden und logische Fehler beim Datenbankentwurf aufzudecken. Sie ersetzen aber nicht den gesunden Menschenverstand, denn es sind Formalismen. Niemand käme auf die Idee, in einer Tabelle die Felder Postleitzahl und Ort auf verschiedene Tabellen zu verteilen, weil sie sonst nicht in der dritten Normalform ist.

Beim Entwurf sollte man die Operationen, die in der geplanten Anwendung auf den Tabellen arbeiten, im Hinterkopf behalten. Es macht einen Unterschied, ob der Datenbestand eher statisch ist und hauptsächlich Leseoperationen zulassen soll oder ob mit vielen Einfüge- und Löschoperationen zu rechnen ist. Im ersten Fall dürfen Einfüge- oder Löschoperationen teurer sein als Leseoperationen. Sie belasten das System aufgrund ihrer geringen Häufigkeit nicht so sehr, während Leseoperationen leistungsfähig sein müssen. Im zweiten Fall ist es genau umgekehrt.

Wenn die Daten aus mehreren Tabellen zusammengeholt werden müssen, ist die Anwendung notwendigerweise weniger leistungsfähig als wenn sie aus einer einzigen Tabelle gelesen werden können. So wird in manchen Fällen eine vollkommene Normalisierung gar nicht wünschenswert sein. Gerade im Internet, wo es darauf ankommt, dass Webseiten möglichst schnell geladen werden sollen, wird man den Fokus eher auf die Effizienz der Anwendung legen als auf einen schulmäßigen Entwurf. Allerdings darf die Integrität der Daten nicht der Effizienz geopfert werden.