O'REILLY®

3. Auflage

Einführung in SQL

Daten erzeugen, bearbeiten und abfragen

Alan Beaulieu
Übersetzung
von Thomas Demmig

3. AUFLAGE

Einführung in SQL

Daten erzeugen, bearbeiten und abfragen

Alan Beaulieu

Deutsche Übersetzung von
Thomas Demmig

O'REILLY®

Alan Beaulieu

Lektorat: Ariane Hesse
Übersetzung: Thomas Demmig
Korrektorat: Sibylle Feldmann, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Karen Montgomery, Michael Oréal, *www.oreal.de*
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-96009-154-7
PDF 978-3-96010-432-2
ePub 978-3-96010-433-9
mobi 978-3-96010-434-6

3. Auflage 2021

Wieblinger Weg 17
69123 Heidelberg

Authorized German translation of the English edition of *Learning SQL, 3rd Edition*
ISBN 978-1-492-05761-1

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«.
O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: kommentar@oreilly.de.

5 4 3 2 1 0

Inhalt

Einleitung

Programmiersprachen kommen und gehen. Nur wenige Sprachen, die heute im Gebrauch sind, haben Wurzeln, die mehr als zehn Jahre zurückreichen. Einige Beispiele sind COBOL, eine Sprache, die immer noch viel in Mainframe-Umgebungen genutzt wird, Java, das Mitte der 1990er-Jahre entstand und zu einer der beliebtesten Programmiersprachen geworden ist, und C, eine Sprache, die nach wie vor für die Entwicklung von Betriebssystemen, Servern und Embedded-Systemen eingesetzt wird. Das im Datenbankbereich gebräuchliche SQL geht sogar bis in die 1970er-Jahre zurück.

SQL wurde ursprünglich dazu geschaffen, Daten aus den seit mehr als 40 Jahren existierenden relationalen Datenbanken anzulegen, zu bearbeiten und abzufragen. In den letzten zehn Jahren haben aber auch andere Plattformen wie Hadoop, Spark oder NoSQL an Fahrt aufgenommen und sich aus dem Markt der relationalen Datenbanken ein Stück vom Kuchen abgeschnitten. Wie wir in den letzten Kapiteln dieses Buchs noch besprechen werden, hat sich die SQL-Sprache trotzdem weiterentwickelt, um Daten von diversen Plattformen zu verarbeiten – unabhängig davon, ob diese in Tabellen, Dokumenten oder einfachen Dateien abgelegt sind.

Wozu SQL lernen?

Egal ob Sie eine relationale Datenbank nutzen werden oder auch nicht – wenn Sie in der Data Science oder einem anderen Bereich der Datenanalyse arbeiten, werden Sie neben weiteren Sprachen und Plattformen wie Python und R sehr wahrscheinlich SQL können müssen. Daten kommen in großen Mengen und sehr schnell von überall her, und Leute, die bedeutsame Informationen aus all diesen Daten extrahieren können, werden immer gesucht.

Warum SQL mit diesem Buch lernen?

Es gibt viele Bücher, die Sie wie Idioten, Dummchen oder etwas Ähnliches behandeln, aber meist kratzen diese nur an der Oberfläche. Am anderen Ende des Spektrums finden Sie Referenzen, die jede Variante jeder Anweisung einer Sprache bis ins Detail durchgehen. Das kann nützlich sein, wenn Sie schon grob wissen, was

Sie tun wollen, und nun nur noch die Syntax dazu benötigen. Dieses Buch möchte einen guten Mittelweg finden – ein paar Hintergründe vorstellen, die Grundlagen der SQL-Sprache behandeln und dann einige der fortgeschritteneren Features behandeln, mit denen Sie glänzen können. Zudem finden Sie am Ende dieses Buchs ein Kapitel zum Abfragen von Daten aus nicht-relationalen Datenbanken, was in einführenden Büchern nur selten ein Thema ist.

Aufbau dieses Buchs

Dieses Buch ist in 18 Kapitel und 2 Anhänge unterteilt:

Kapitel 1, Der Hintergrund
: geht auf die Geschichte der Computerdatenbanken ein und schildert den Aufstieg des relationalen Modells und der Sprache SQL.

Kapitel 2, Datenbanken erstellen und mit Daten füllen
: zeigt, wie man eine MySQL-Datenbank anlegt, die Tabellen für die Beispiele dieses Buchs erstellt und Daten in diese Tabellen lädt.

Kapitel 3, Datenbankabfragen
: führt die `select`-Anweisung ein und stellt die gebräuchlichsten Klauseln vor (`select`, `from`, `where`).

Kapitel 4, Filtern
: beschreibt die verschiedenen Arten von Bedingungen, die in der `where`-Klausel einer `select`-, `update`- oder `delete`-Anweisung verwendet werden können.

Kapitel 5, Mehrere Tabellen abfragen
: zeigt, wie man mehrere Tabellen mittels Tabellen-Joins benutzen kann.

Kapitel 6, Umgang mit Mengen
: handelt von Datenmengen und der Frage, wie diese innerhalb von Abfragen interagieren.

Kapitel 7, Daten erzeugen, bearbeiten und konvertieren
: stellt verschiedene eingebaute Funktionen vor, mit denen man Daten bearbeiten oder konvertieren kann.

Kapitel 8, Gruppieren und Aggregieren von Daten
: zeigt, wie Daten zusammengefasst werden.

Kapitel 9, Unterabfragen
: führt Unterabfragen ein (ich liebe Unterabfragen) und zeigt, wie und wo man sie einsetzen kann.

Kapitel 10, Weitere Joins
: geht genauer auf die verschiedenen Join-Typen ein.

Kapitel 11, Bedingungslogik
: erklärt, wie man Bedingungslogik (d. h. `if-then-else`) in `select`-, `insert`-, `update`- und `delete`-Anweisungen verwendet.

Kapitel 12, Transaktionen
: führt Transaktionen ein und zeigt, wie man sie nutzt.

Kapitel 13, Indizes und Constraints
: erklärt Indizes und Constraints.

Kapitel 14, Views
: zeigt, wie man eine Schnittstelle aufbaut, die Datenkomplexität vor Benutzern verbirgt.

Kapitel 15, Metadaten
: zeigt den Nutzen des Data Dictionary.

Kapitel 16, Analytische Funktionen
: behandelt Funktionalität zum Erzeugen von Rankings, Untersummen und anderen Werten, die im Reporting und in der Analyse oft zum Einsatz kommen.

Kapitel 17, Mit großen Datenbanken arbeiten
: zeigt Techniken zum einfacheren Managen und Arbeiten mit sehr großen Datenbanken.

Kapitel 18, SQL und Big Data
: untersucht die Anpassungen der SQL-Sprache, mit denen Daten aus nichtrelationalen Datenplattformen abgefragt werden können.

Anhang A, ER-Diagramm der Musterdatenbank
: zeigt das Datenbankschema, das für alle Beispiele dieses Buchs verwendet wird.

Anhang B, Lösungen zu den Übungen
: enthält die Lösungen der Übungsaufgaben.

Verwendete Konventionen

In diesem Buch gelten folgende typografische Konventionen:

Kursiv
: Für neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.

`Nichtproportionalschrift`
: Wird für Codebeispiele genutzt, aber auch im Text, um Programmelemente wie Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter zu kennzeichnen.

`Nichtproportionalschrift kursiv`
: Wird für Text verwendet, den der Benutzer individuell eingeben muss oder der sich aus dem Kontext ergibt.

`Nichtproportionalschrift fett`
: Befehle oder anderer Text, der vom Benutzer so eingegeben werden soll, oder Stellen im Code, die besonders hervorgehoben werden sollen.

Ein Tipp, Vorschlag oder allgemeiner Hinweis. Ich weise in solchen Einschüben beispielsweise auf neue Features von Datenbanken hin.

Eine Warnung. So werde ich zum Beispiel darauf aufmerksam machen, dass eine bestimmte SQL-Klausel unerwünschte Folgen haben kann, wenn man sie nicht vorsichtig genug einsetzt.

Verwendung der Codebeispiele

Um mit den Daten zu experimentieren, die für die Beispiele in diesem Buch genutzt wurden, haben Sie zwei Möglichkeiten:

- Laden Sie die MySQL-Server-Version 8.0 (oder neuer) herunter und installieren Sie sie. Holen Sie sich dann die Sakila-Beispieldatenbank, die Sie unter *https://dev.mysql.com/doc/index-other.html* finden.
- Rufen Sie *https://www.katacoda.com/mysql-db-sandbox/scenarios/mysql-sandbox* auf, um auf die MySQL-Sandbox zuzugreifen, in der die Sakila-Beispieldatenbank in einer MySQL-Instanz geladen ist. Sie müssen dafür einen (kostenlosen) Katacoda-Account einrichten. Dann klicken Sie auf den Button *Start Scenario*.

Haben Sie die zweite Möglichkeit gewählt, wird nach dem Starten des Szenarios ein MySQL-Server installiert und gestartet, und anschließend wird das Sakila-Schema mit seinen Daten geladen. Ist alles erledigt, erscheint ein Standardprompt `mysql>`, und Sie können damit beginnen, die Beispieldatenbank abzufragen. Das ist die einfachste Option, und ich vermute, die meisten Leser werden sie wählen – wenn das für Sie gut genug klingt, können Sie einfach zum nächsten Abschnitt springen.

Möchten Sie lieber Ihre eigene Kopie der Daten nutzen und Änderungen daran dauerhaft machen, oder sind Sie einfach daran interessiert, den MySQL-Server auf Ihrem eigenen Rechner zu installieren, bevorzugen Sie vielleicht die erste Option. Sie können sich auch dazu entscheiden, einen MySQL-Server zu nutzen, der in der Cloud gehostet wird, wie zum Beispiel bei Amazon Web Services oder Google Cloud. Sie müssen dann auf jeden Fall die Installation und die Konfiguration selbst vornehmen, was aber nicht Bestandteil dieses Buchs ist. Ist Ihre Datenbank bereit, werden Sie noch ein paar weitere Schritte ausführen müssen, um die Sakila-Beispieldatenbank zu laden.

Als Erstes müssen Sie das `mysql`-Kommandozeilentool starten und ein Passwort angeben, danach führen Sie die folgenden Schritte aus:

1. Öffnen Sie *https://dev.mysql.com/doc/index-other.html* und laden Sie die Dateien für *sakila database* aus dem Abschnitt *Example Databases* herunter.
2. Legen Sie sie in einem lokalen Verzeichnis wie *C:\temp\sakila-db* ab (das nutzen wir für die nächsten beiden Schritte, aber Sie können natürlich gern einen anderen Pfad wählen).
3. Geben Sie `source c:\temp\sakila-db\sakila-schema.sql;` ein und drücken Sie die Taste Enter.
4. Geben Sie `source c:\temp\sakila-db\sakila-data.sql;` ein und drücken Sie die Taste Enter.

Jetzt sollten Sie eine laufende Datenbank haben, die mit all den Daten gefüllt ist, die Sie für die Beispiele in diesem Buch benötigen.

Danksagungen

Ich möchte meinem Lektor Jeff Bleiel dafür danken, dass er mir dabei geholfen hat, diese dritte Auflage Wirklichkeit werden zu lassen. Vielen Dank auch an Thomas Nield, Ann White-Watkins und Charles Givre, die so nett waren, das Buch für mich durchzusehen. Außerdem möchte ich mich bei Deb Baker, Jess Haberman und all den anderen Leuten bei O'Reilly Media bedanken, die an diesem Buch beteiligt waren. Schließlich möchte ich meiner Frau Nancy und meinen Töchtern Michelle und Nicole für ihre Ermutigung und Inspiration danken.

KAPITEL 1

Der Hintergrund

Ehe wir nun die Ärmel aufkrempeln und uns in die Arbeit stürzen, wäre es sicherlich hilfreich, die Geschichte der Datenbanktechnologie zu umreißen, um besser zu verstehen, wie sich relationale Datenbanken und die SQL-Sprache entwickelt haben. Daher möchte ich zunächst einige Grundkonzepte vorstellen und die Geschichte der computergestützten Datenspeicherung und -abfrage anschauen.

Wer möglichst schnell damit beginnen möchte, Queries zu schreiben, kann gerne direkt zu Kapitel 3 springen, aber ich empfehle, später noch einmal die ersten beiden Kapitel anzuschauen, um die Geschichte und den Einsatz von SQL besser zu verstehen.

Einführung in Datenbanken

Eine *Datenbank* ist im Grunde nichts weiter als eine Menge von zusammenhängenden Informationen. So ist zum Beispiel ein Telefonbuch eine Datenbank mit den Namen, Telefonnummern und Adressen aller Bewohner einer bestimmten Gegend. Doch so ein Telefonbuch ist zwar überall zu finden und wird viel genutzt, aber es hat auch einige Mängel:

- Wegen der Vielzahl der Einträge kann es zeitraubend sein, die Telefonnummer eines bestimmten Teilnehmers darin zu finden.
- Der einzige Index eines Telefonbuchs ist die alphabetische Ordnung nach Nachname und Vorname. Eine Möglichkeit, die Namen der Menschen mit einer bestimmten Adresse herauszufinden, gibt es nur in der Theorie.
- Von dem Augenblick an, da das Telefonbuch in Druck geht, fängt es bereits an zu veralten: Menschen ziehen fort, wechseln ihre Telefonnummer oder gehen an eine andere Adresse in derselben Gegend.

Diese Nachteile hat jedes manuelle System der Datenspeicherung, also beispielsweise auch Patientendaten, die in einem Aktenschrank abgelegt sind. Da eine papiergebundene Datensammlung eine derart sperrige Angelegenheit ist, gehörten *Datenbanksysteme* zu den ersten Computeranwendungen überhaupt. In ihnen werden Daten computergestützt gespeichert und abgefragt. Da ein Datenbanksystem diese Arbeiten nicht auf Papier, sondern elektronisch leistet, kann es die Daten

schneller abfragen, mit unterschiedlichen Indizes versehen und seinen Benutzern immer die aktuellsten Informationen liefern.

Die frühen Datenbanksysteme verwalteten die gespeicherten Daten noch auf Magnetbändern. Weil es im Allgemeinen viel mehr Bänder als Bandlesegeräte gab, mussten permanent Techniker Bänder austauschen, wenn bestimmte Informationen angefordert wurden. Und da die Computer jener Zeit noch sehr wenig Arbeitsspeicher hatten, mussten dieselben Daten, wenn sie mehrmals angefragt wurden, auch mehrmals von den Bändern gelesen werden. Selbst wenn diese Datenbanksysteme bereits viel besser als die papiergebundenen waren, so waren sie doch weit von dem entfernt, was mit unserer heutigen Technologie möglich ist. (Moderne Datenbanksysteme können Petabytes von Daten verwalten, die über eine Vielzahl von Serverclustern verteilt sind, und sie können zig Gigabytes an Daten in ihren Hochleistungsarbeitsspeichern cachen. Aber ich greife vor.)

Nicht-relationale Datenbanksysteme

Dieser Abschnitt enthält einige Hintergrundinformationen zu Datenbanksystemen, die den eigentlichen relationalen Datenbanksystemen vorausgingen. Wenn Sie es eilig haben und sich sofort in SQL stürzen wollen, können Sie die folgenden Seiten bis zum nächsten Abschnitt gern überspringen.

In den ersten Dekaden des Computerzeitalters wurden Datenbankdaten auf diverse Arten gespeichert und dargestellt. In einem *hierarchischen Datenbanksystem* werden sie beispielsweise in Baumstrukturen angeordnet. Abbildung 1-1 zeigt, wie die Daten der Bankkonten von George Blake und Sue Smith als Baumstruktur wiedergegeben werden könnten.

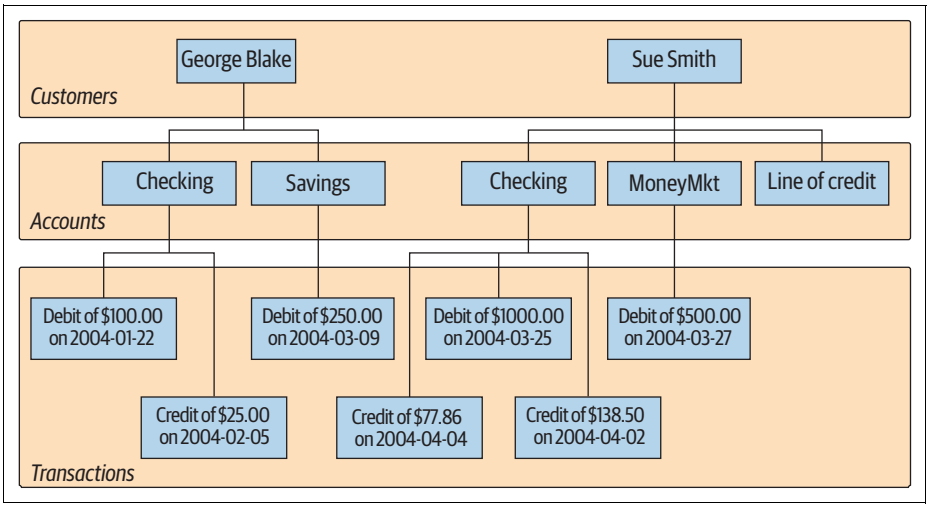


Abbildung 1-1: Hierarchische Sicht der Kontodaten

George und Sue haben jeweils einen eigenen Baum, in dem ihre Konten und Kontobewegungen gespeichert sind. Das hierarchische Datenbanksystem bietet Mittel und Wege, um den Baum eines bestimmten Bankkunden ausfindig zu machen und ihn dann nach den gewünschten Konten und/oder Kontobewegungen zu durchforsten. Jeder Knoten im Baum hat null oder einen Elternknoten und null, ein oder mehrere Kinder. Diese Konfiguration bezeichnet man als *Single-Parent-Hierarchie*.

Ein anderes beliebtes Verfahren namens *Netzwerkdatenbanksystem* stellt Datensätze und Verknüpfungsmengen dar, um die Beziehungen zwischen den verschiedenen Datensätzen zu definieren. Abbildung 1-2 zeigt, wie dieselben Bankkonten von George und Sue in einem solchen System aussehen könnten.

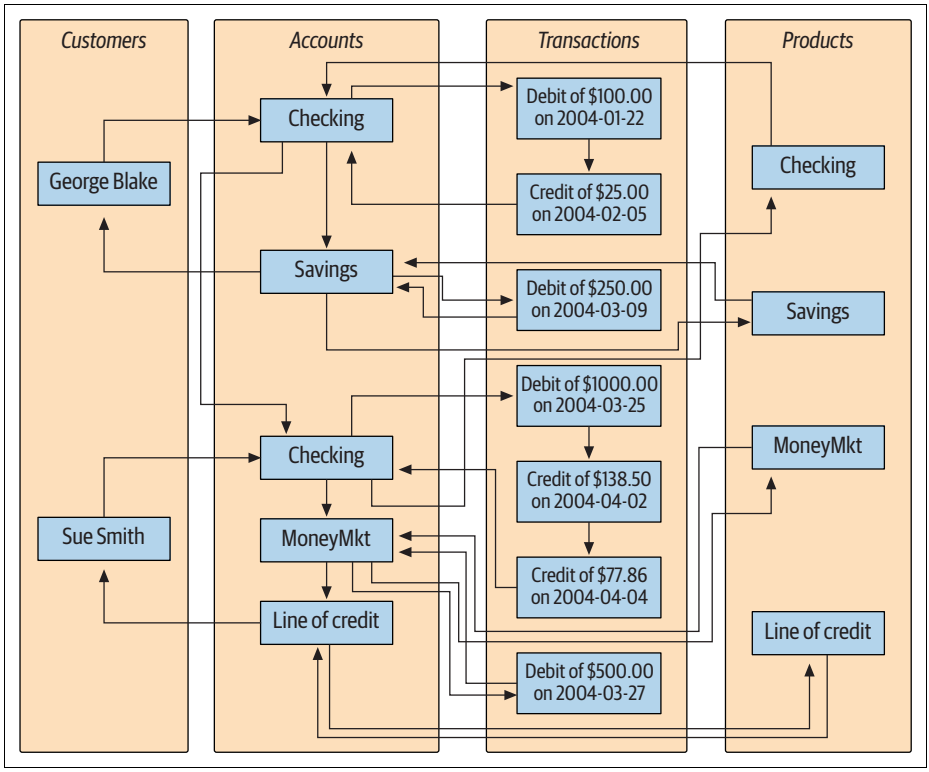


Abbildung 1-2: Netzwerksicht der Kontodaten

Um die Bewegungen von Sues Tagesgeldkonto wiederzufinden, wären folgende Schritte vonnöten:

1. Den Kundendatensatz von Sue Smith finden.
2. Der Verknüpfung von diesem Datensatz zu Sues Kontenliste folgen.
3. Die Konten durchgehen, bis das Tagesgeldkonto gefunden ist.
4. Der Verknüpfung zwischen dem Tagesgeldkonto und seiner Transaktionsliste folgen.

Ein interessantes Merkmal der Netzwerkdatenbanksysteme wird an der Menge der `product`-Einträge ganz rechts in Abbildung 1-2 erkennbar. Beachten Sie, dass jeder `product`-Eintrag (*Checking*, *Savings* usw.) auf eine Liste von `account`-Einträgen verweist, die den gleichen Product-Typ haben. Dadurch kann man von verschiedenen Orten aus auf `account`-Einträge zugreifen (nämlich sowohl von den `customer`- als auch von den `product`-Datensätzen aus). So wird eine Netzwerkdatenbank zur *Multi-Parent-Hierarchie*.

Sowohl hierarchische als auch Netzwerkdatenbanksysteme sind auch heutzutage noch quicklebendig, wenngleich hauptsächlich im Mainframe-Umfeld. Außerdem erlebten hierarchische Datenbanksysteme im Bereich der Verzeichnisdienste eine Renaissance, also zum Beispiel im Active Directory von Microsoft und im Open Source Directory Server von Apache. Doch Anfang der 1970er-Jahre kam eine neue Art der Datendarstellung auf, die strenger, aber zugleich auch einfacher zu verstehen und zu implementieren war.

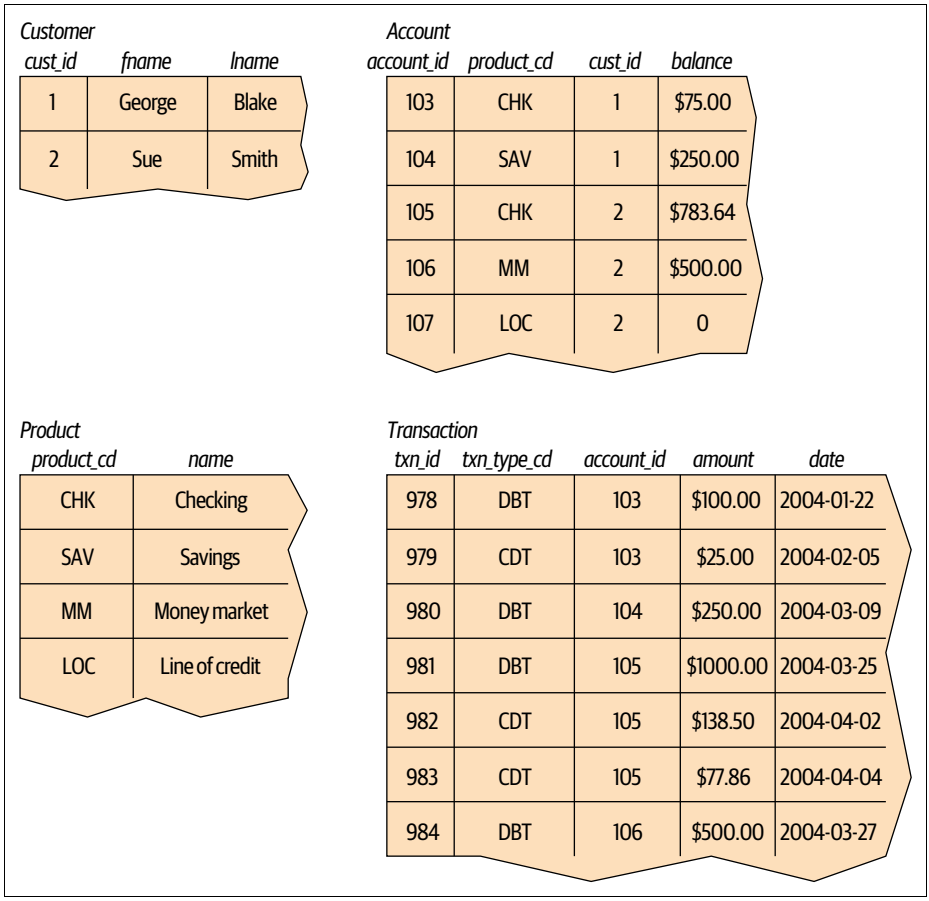
Das relationale Modell

Im Jahr 1970 veröffentlichte Dr. E. F. Codd vom IBM-Forschungslabor ein Paper mit dem Titel »A Relational Model of Data for Large Shared Data Banks« (»Ein relationales Datenmodell für große, verteilte Datenbanken«), in dem er vorschlug, Daten als Mengen von *Tabellen* darzustellen. Anstatt mithilfe von Pointern zwischen verwandten Entitäten zu navigieren, verwendet dieses System redundante Daten, um Datensätze miteinander zu verknüpfen, die in verschiedenen Tabellen vorliegen. Abbildung 1-3 zeigt, wie die Kontodaten von George und Sue in einem solchen Kontext aussehen würden.

Abbildung 1-3 zeigt vier Tabellen, um die vier bisher verwendeten Entitäten darzustellen: `customer`, `product`, `account` und `transaction`. Wenn Sie einen Blick auf den Anfang der `customer`-Tabelle in Abbildung 1-3 werfen, erkennen Sie drei *Spalten*: `cust_id` (mit der Kundennummer), `fname` (mit dem Vornamen des Kunden) und `lname` (mit dem Nachnamen des Kunden). Außerdem hat die `customer`-Tabelle zwei *Zeilen*, eine mit den Daten von George Blake und eine mit den Daten von Sue Smith. Wie viele Spalten eine Tabelle höchstens haben darf, ist vom Server abhängig, doch die Zahl ist normalerweise groß genug (Microsoft SQL Server gestattet zum Beispiel 1.024 Spalten pro Tabelle). Die Höchstzahl der Zeilen einer Tabelle ist eher eine Frage der physikalischen Grenzen (d.h. des verfügbaren Plattenplatzes) und der Wartbarkeit (d.h., wie umfangreich eine Tabelle werden kann, bevor die Arbeit mit ihr zu kompliziert wird) als eine Frage der Serverlimits.

Jede Tabelle in einer relationalen Datenbank enthält Informationen, um eine Tabellenzeile eindeutig zu identifizieren (den sogenannten *Primärschlüssel*), sowie weitere Informationen, die benötigt werden, um die dargestellte Entität vollständig zu beschreiben. Wenn Sie noch einmal die `customer`-Tabelle anschauen, sehen Sie, dass die `cust_id`-Spalte für jeden Kunden eine andere Nummer speichert. So ist etwa George Blake durch seine Kundennummer 1 eindeutig identifiziert. Kein an-

derer Kunde wird jemals diese Kennung bekommen, und es sind keine anderen Informationen erforderlich, um George Blakes Daten in der `customer`-Tabelle wiederzufinden.



Customer

cust_id	*fname*	*lname*
1	George	Blake
2	Sue	Smith

Account

account_id	*product_cd*	*cust_id*	*balance*
103	CHK	1	$75.00
104	SAV	1	$250.00
105	CHK	2	$783.64
106	MM	2	$500.00
107	LOC	2	0

Product

product_cd	*name*
CHK	Checking
SAV	Savings
MM	Money market
LOC	Line of credit

Transaction

txn_id	*txn_type_cd*	*account_id*	*amount*	*date*
978	DBT	103	$100.00	2004-01-22
979	CDT	103	$25.00	2004-02-05
980	DBT	104	$250.00	2004-03-09
981	DBT	105	$1000.00	2004-03-25
982	CDT	105	$138.50	2004-04-02
983	CDT	105	$77.86	2004-04-04
984	DBT	106	$500.00	2004-03-27

Abbildung 1-3: Relationale Sicht der Kontodaten

Jeder Datenbankserver bietet einen Mechanismus zur Erstellung eindeutiger Zahlenmengen, die als Primärschlüsselwerte eingesetzt werden können. Sie müssen sich also nicht selbst darum kümmern, welche Zahlen bereits vergeben wurden.

Zwar hätte ich auch eine Kombination aus den Spalten `fname` und `lname` als Primärschlüssel auswählen können (einen Primärschlüssel, der aus mehreren Spalten besteht, bezeichnet man als *zusammengesetzten Schlüssel*), doch es wäre gut möglich, dass mehrere Bankkunden den gleichen Vor- und Nachnamen haben. Daher beschloss ich, extra die Spalte `cust_id` in die `customer`-Tabelle einzuführen, um sie als Primärschlüsselspalte zu verwenden.

Hätte man in diesem Beispiel `fname`/`lname` als Primärschlüssel gewählt, würde man das als einen *sprechenden Schlüssel* bezeichnen, während `cust_id` als Primärschlüssel *Surrogatschlüssel* genannt wird. Die Entscheidung darüber, ob man sprechende Schlüssel oder Surrogatschlüssel einsetzen sollte, ist Sache des Datenbankdesigners, aber in diesem speziellen Fall ist die Wahl eindeutig, da sich der Nachname einer Person ändern kann (beispielsweise weil der Nachname des Ehepartners übernommen wird) und sich Primärschlüssel niemals ändern sollten, nachdem der Wert einmal zugewiesen wurde.

Manche der Tabellen enthalten auch Informationen, um zu einer anderen Tabelle zu navigieren; das ist der Punkt, an dem die zuvor erwähnten »redundanten Daten« ins Spiel kommen. So enthält beispielsweise die `account`-Tabelle eine Spalte namens `cust_id` mit der eindeutigen Kennung des Kunden, der das Konto eröffnet hat, sowie eine Spalte namens `product_cd` mit der eindeutigen Kennung des Produkts, dem das Konto entspricht. Diese sogenannten *Fremdschlüssel* haben denselben Zweck wie die Linien, mit denen die Entitäten in der hierarchischen und der Netzwerkdarstellung der Kontodaten verbunden sind. Wenn Sie einen bestimmten Datensatz mit Kontodaten betrachten und mehr Informationen über den Kunden haben möchten, der das Konto geöffnet hat, würden Sie den Wert der Spalte `cust_id` einsetzen, um die entsprechende Zeile in der Tabelle `customer` zu finden (in Fachjargon bezeichnet man einen solchen Vorgang als *Join*; Joins werden in Kapitel 3 eingeführt und in Kapitel 5 und Kapitel 10 ausführlich betrachtet).

Zunächst mag es nach Verschwendung aussehen, dieselben Daten viele Male zu speichern, aber das relationale Modell sagt ziemlich klar aus, welche redundanten Daten gespeichert werden können. So kann zum Beispiel die `account`-Tabelle sehr wohl eine Spalte für die Kundennummer des Kontoinhabers haben, aber nicht für seinen Vor- und Nachnamen. Wenn beispielsweise ein Kunde seinen Namen wechselt, möchte man sicher sein, dass dieser Name nur an einer einzigen Stelle der Datenbank gespeichert ist, sonst kann es passieren, dass die Daten nicht überall aktualisiert werden und somit inkonsistent werden. Der einzig richtige Platz für diese Daten ist die `customer`-Tabelle, und nur die `cust_id`-Werte sollten in anderen Tabellen verwendet werden. Außerdem darf eine einzelne Spalte nicht mehrere Daten enthalten; es darf also nicht einfach eine `name`-Spalte geben, die sowohl den Vor- als auch den Nachnamen des Kunden enthält, oder eine `address`-Spalte, in der Straße, Ort, Staat und Postleitzahl auf einmal stehen. Wird ein Datenbankentwurf so weit verfeinert, dass jede einzelne Information an genau einer Stelle vertreten ist (abgesehen von Fremdschlüsseln), bezeichnet man dies als *Normalisierung*.

Vielleicht fragen Sie sich, wie man in den vier Tabellen in Abbildung 1-3 die Kontobewegungen des Girokontos von George Blake wiederfinden kann. Zuerst suchen Sie sich die Kundennummer von George Blake in der `customer`-Tabelle heraus. Dann finden Sie in der `account`-Tabelle die Zeile, deren `cust_id`-Spalte Georges Kundennummer enthält und deren `product_cd`-Spalte mit der Zeile der `product`-Tabelle übereinstimmt, deren `name`-Spalte den Eintrag *Checking* aufweist. Zum Schluss ma-

chen Sie dann die Zeilen der `transaction`-Tabelle ausfindig, deren `account_id`-Spalte wiederum die eindeutige ID der `account`-Tabelle enthält. Das mag vielleicht kompliziert klingen, kann aber mit SQL in einer einzigen Anweisung durchgeführt werden, wie Sie gleich noch sehen werden.

Ein wenig Fachjargon

Da ich bereits in den vorangegangenen Abschnitten einige neue Fachbegriffe eingeführt habe, ist es nun an der Zeit für ein paar formale Definitionen. Tabelle 1-1 zeigt die Begriffe, die im Rest dieses Buchs immer wieder verwendet werden, zusammen mit ihren Definitionen.

Tabelle 1-1: Fachbegriffe und Definitionen

Begriff	Definition
Entität	Etwas, das für die Nutzer der Datenbank von Interesse ist, zum Beispiel Kunden, Teile, Orte usw.
Spalte	Eine einzelne Information, die in einer Tabelle gespeichert ist.
Zeile	Eine Menge von Spalten, die zusammen eine Entität oder einen Vorgang einer Entität vollständig beschreiben. Wird auch als Datensatz oder Eintrag bezeichnet.
Tabelle	Eine Menge von Zeilen, die entweder im Arbeitsspeicher (nicht-persistent) oder in einem dauerhaften Speicher (persistent) gespeichert sind.
Ergebnismenge	Ein anderer Name für eine nicht-persistente Tabelle, im Allgemeinen das Ergebnis einer SQL-Abfrage.
Primärschlüssel	Eine oder mehrere Spalten, die als eindeutiger Identifier für jede Zeile der Tabelle dienen.
Fremdschlüssel	Eine oder mehrere Spalten, die zusammengenommen eine einzelne Zeile in einer anderen Tabelle identifizieren.

Was ist SQL?

Zusammen mit der Definition des relationalen Modells stellte Codd eine Sprache namens DSL/Alpha vor, um die Daten in relationalen Tabellen zu bearbeiten. Kurz nach der Veröffentlichung von Codds Paper stellte IBM eine Arbeitsgruppe für den Bau eines Prototyps zusammen, der auf Codds Ideen basieren sollte. Diese Gruppe erschuf eine vereinfachte Version von DSL/Alpha namens SQUARE. Durch weitere Verfeinerungen an SQUARE entstand eine Sprache namens SEQUEL, die dann schließlich in SQL umbenannt wurde. Während die Sprache zunächst dazu diente, Daten in relationalen Datenbanken zu verarbeiten, hat sie sich mittlerweile zu einer Sprache gemausert (wie Sie im hinteren Teil des Buchs noch sehen werden), mit der auf Daten in verschiedensten Datenbanktechnologien zugegriffen werden kann.

SQL ist mittlerweile über 40 Jahre alt und hat sich mit der Zeit massiv gewandelt. Mitte der 1980er-Jahre begann das *American National Standards Institute* (ANSI), den ersten Standard für SQL auszuarbeiten, der 1986 veröffentlicht wurde. Wei-

tere Verfeinerungen führten zu neuen Releases des SQL-Standards in den Jahren 1989, 1992, 1999, 2003, 2006, 2008, 2011 und 2016. Doch nicht nur der Sprachkern wurde überarbeitet, SQL erhielt auch neue Features, um beispielsweise objektorientierte Funktionalität zu unterstützen. Die jüngeren Standards konzentrieren sich auf die Integration passender Technologien, wie zum Beispiel von XML (*Extensible Markup Language*) und JSON (*JavaScript Object Notation*).

SQL geht mit dem relationalen Modell Hand in Hand, da das Ergebnis einer SQL-Abfrage immer eine Tabelle ist (auch wenn es in diesem Kontext *Ergebnismenge* heißt). So kann eine neue permanente Tabelle in einer relationalen Datenbank einfach schon durch die Speicherung der Ergebnismenge einer Abfrage angelegt werden. Zudem kann eine Abfrage sowohl permanente Tabellen als auch die Ergebnismenge anderer Abfragen als Eingabe nutzen (dies wird in Kapitel 9 noch genauer erläutert).

Ein Hinweis zum Schluss: SQL ist kein Akronym (obwohl manche darauf bestehen, es sei die Abkürzung für »Structured Query Language«). Sie können den Namen sowohl als einzelne Buchstaben aussprechen (S. Q. L.) als auch wie das englische Wort »Sequel«.

SQL-Anweisungen

SQL besteht aus mehreren getrennten Teilen, von denen folgende in diesem Buch behandelt werden: SQL-Schemaanweisungen, mit denen die in der Datenbank gespeicherten Datenstrukturen definiert werden, SQL-Datenanweisungen, mit denen die zuvor durch SQL-Schemaanweisungen angelegten Datenstrukturen bearbeitet werden, und SQL-Transaktionsanweisungen, mit denen Transaktionen gestartet, beendet oder zurückgerollt werden können (siehe Kapitel 12). Wenn Sie zum Beispiel eine neue Tabelle in Ihrer Datenbank anlegen möchten, verwenden Sie dazu die SQL-Schemaanweisung `create table`; um jedoch diese neue Tabelle mit Daten zu bevölkern, verwenden Sie die SQL-Datenanweisung `insert`.

Um Ihnen einen Vorgeschmack dieser Anweisungen zu geben, sehen Sie hier eine SQL-Schemaanweisung, mit der die Tabelle `corporation` angelegt wird:

```
CREATE TABLE corporation
 (corp_id SMALLINT,
  name VARCHAR(30),
  CONSTRAINT pk_corporation PRIMARY KEY (corp_id)
 );
```

Diese Anweisung erstellt eine Tabelle mit den beiden Spalten `corp_id` und `name`, wobei die `corp_id` der Primärschlüssel ist. Die genaueren Einzelheiten dieser Anweisung, wie zum Beispiel die verschiedenen für MySQL verfügbaren Datentypen, werden im nächsten Kapitel erläutert. Als Nächstes sehen Sie hier eine SQL-Datenanweisung, mit der eine Zeile für die Acme Paper Corporation in die Tabelle `corporation` eingefügt wird:

```
INSERT INTO corporation (corp_id, name)
VALUES (27, 'Acme Paper Corporation');
```

Diese Anweisung schreibt in die `corporation`-Tabelle eine Zeile mit dem Wert `27` als `corp_id` und dem Wert `Acme Paper Corporation` für die `name`-Spalte.

Abschließend folgt noch eine einfache `select`-Anweisung, mit der die soeben angelegten Daten abgefragt werden:

```
mysql< SELECT name
    -> FROM corporation
    -> WHERE corp_id = 27;
+------------------------+
| name                   |
+------------------------+
| Acme Paper Corporation |
+------------------------+
```

Alle durch SQL-Schemaanweisungen erstellten Datenbankelemente werden in einem speziellen Tabellensatz gespeichert, den man *Data Dictionary* nennt. Diese »Daten über die Datenbank« bezeichnet man zusammengenommen als *Metadaten*. Wir werden sie in Kapitel 15 näher beleuchten. Genau wie die von Ihnen selbst angelegten Tabellen können Sie auch die Data-Dictionary-Tabellen mit einer `select`-Anweisung abfragen. So können Sie die aktuellen Datenstrukturen erkennen, die in der Datenbank zur Laufzeit eingesetzt werden. Wenn man Ihnen beispielsweise aufträgt, einen Report zu erstellen, der die im letzten Monat neu angelegten Konten zeigt, können Sie entweder die Namen der Spalten der `account`-Tabelle hartcodieren, die Ihnen beim Schreiben des Reports bekannt waren, oder Sie können das Data Dictionary abfragen, um festzustellen, welche Spalten aktuell vorhanden sind, und den Report jedes Mal dynamisch generieren.

Ein Großteil dieses Buchs befasst sich mit dem datenorientierten Teil von SQL, der aus den Anweisungen `select`, `update`, `insert` und `delete` besteht. SQL-Schemaanweisungen sehen Sie in Kapitel 2: Dort begleite ich Sie dabei, ein paar einfache Tabellen zu entwerfen und anzulegen. Über SQL-Schemaanweisungen gibt es, abgesehen von ihrer Syntax, normalerweise nicht viel zu sagen, während die SQL-Datenanweisungen trotz ihrer kleinen Zahl eine Fülle von Möglichkeiten bieten, die zu untersuchen sich lohnt. Deswegen werden sich die meisten Kapitel in diesem Buch – obwohl ich versuchen werde, Ihnen viele der SQL-Schemaanweisungen vorzustellen – auf die SQL-Datenanweisungen konzentrieren.

SQL: eine nicht-prozedurale Sprache

Wenn Sie in der Vergangenheit bereits mit Programmiersprachen gearbeitet haben, sind Sie den Umgang mit Variablen, Datenstrukturen, Bedingungslogik (`if-then-else`) und Schleifenkonstrukten (`do while ... end`) gewohnt und wissen, wie man Code in kleine, wiederverwendbare Module (Objekte, Funktionen, Prozeduren) zerlegt. Sie übergeben Ihren Code an einen Compiler, und die kompilierte, aus-

führbare Datei tut genau das, was Sie in Ihrem Programm bezweckt haben (nun ja, vielleicht nicht immer *genau*). Wenn Sie mit Java, Python, Scala oder einer anderen *prozeduralen* Sprache arbeiten, haben Sie alles, was das Programm tut, genau unter Kontrolle.

Eine prozedurale Sprache definiert sowohl die gewünschten Ergebnisse als auch den Mechanismus oder Prozess, mit dem die Ergebnisse generiert werden. Nicht-prozedurale Sprachen definieren zwar ebenfalls die gewünschten Ergebnisse, überlassen den Prozess, über den diese Ergebnisse generiert werden, aber einer externen Instanz.

Doch mit SQL müssen Sie einen Teil dieser Kontrolle abgeben, da SQL-Anweisungen zwar die notwendigen Ein- und Ausgaben definieren, die Art und Weise aber, wie eine Anweisung ausgeführt wird, wird von einer Komponente Ihrer Datenbank-Engine bestimmt: der sogenannten *Optimierung*. Sie hat die Aufgabe, Ihre SQL-Anweisungen genau zu betrachten und unter Berücksichtigung der Tabellenkonfiguration und der verfügbaren Indizes den effizientesten Ausführungspfad auszutüfteln (nun ja, nicht immer den allereffizientesten). Die meisten Datenbank-Engines ermöglichen es, die Entscheidungen der Optimierung durch *Optimierungshinweise* (*Optimizer Hints*) zu beeinflussen, in denen beispielsweise gesagt wird, dass ein bestimmter Index benutzt werden soll. Ein Großteil der normalen SQL-Benutzer schwingt sich jedoch niemals zu diesen luftigen Höhen auf und überlässt solche Hacks Datenbankadministratoren oder Performanceexperten.

Mit SQL kann man also keine vollständigen Anwendungen schreiben. Entweder schreiben Sie ein einfaches Skript, um bestimmte Daten zu bearbeiten, oder Sie integrieren SQL in Ihre bevorzugte Programmiersprache. Einige Datenbankhersteller haben Ihnen diese Arbeit bereits abgenommen, wie Oracle mit der Programmiersprache PL/SQL, MySQL mit seiner Sprache für gespeicherte Routinen und Microsoft mit der Sprache Transact-SQL. Bei diesen Sprachen sind SQL-Datenanweisungen Teil der Grammatik, was eine nahtlose Integration von Datenbankabfragen in prozedurale Befehle ermöglicht. Wenn Sie keine spezielle Datenbanksprache verwenden, also etwa mit Java oder Python arbeiten, benötigen Sie ein Toolkit oder eine API, um SQL-Anweisungen in Ihrem Code auszuführen. Manche derartigen Toolkits werden vom Datenbankhersteller, andere von Drittanbietern oder Open-Source-Providern zur Verfügung gestellt. Tabelle 1-2 zeigt einige Möglichkeiten, um SQL in eine spezifische Sprache zu integrieren.

Tabelle 1-2: Toolkits zur SQL-Integration

Sprache	Toolkit
Java	JDBC (Java Database Connectivity)
C#	ADO.NET (Microsoft)
Ruby	Ruby DBI

Tabelle 1-2: Toolkits zur SQL-Integration (Fortsetzung)

Sprache	Toolkit
Python	Python DB
Go	Paket database/sql

Wenn Sie lediglich SQL-Befehle interaktiv ausführen möchten, finden Sie bei jedem Datenbankhersteller mindestens ein einfaches Kommandozeilentool, um SQL-Befehle an die Datenbank-Engine zu übermitteln und die Ergebnisse zu inspizieren. Die meisten Hersteller liefern auch eine grafische Oberfläche, die in einem Fenster Ihre SQL-Befehle und in einem anderen die Ergebnisse dieser Befehle zeigt. Zudem gibt es Tools von Drittanbietern, wie zum Beispiel SQuirrel, das sich über eine JDBC-Verbindung mit vielen verschiedenen Datenbankservern verbindet. Da die Beispiele in diesem Buch an einer MySQL-Datenbank ausprobiert werden, verwende ich das Kommandozeilentool `mysql`, das Teil jeder MySQL-Installation ist, um den Code auszuführen und die Ergebnisse zu formatieren.

SQL-Beispiele

Weiter oben in diesem Kapitel hatte ich Ihnen versprochen, eine SQL-Anweisung zu zeigen, die alle Bewegungen auf George Blakes Girokonto zurückliefert. Hier ist sie:

```
SELECT t.txn_id, t.txn_type_cd, t.txn_date, t.amount
FROM individual i
  INNER JOIN account a ON i.cust_id = a.cust_id
  INNER JOIN product p ON p.product_cd = a.product_cd
  INNER JOIN transaction t ON t.account_id = a.account_id
WHERE i.fname = 'George' AND i.lname = 'Blake'
  AND p.name = 'checking account';
+--------+-------------+---------------------+--------+
| txn_id | txn_type_cd | txn_date            | amount |
+--------+-------------+---------------------+--------+
|     11 | DBT         | 2008-01-05 00:00:00 | 100.00 |
+--------+-------------+---------------------+--------+
1 row in set (0.00 sec)
```

Ohne an diesem Punkt gleich zu sehr ins Detail zu gehen: Diese Abfrage identifiziert in der Tabelle `individual` die Zeile für George Blake und in der Tabelle `product` die Zeile für das »checking«-Produkt, sucht dann in der Tabelle `account` die Zeile für diese spezielle Person-Produkt-Kombination und liefert vier Zeilen aus der Tabelle `transaction` mit allen für dieses Konto gemeldeten Transaktionen. Wenn Sie zufällig wissen, dass die Kundennummer von George Blake 8 ist und die entsprechende Kontoart über den Code `'CHK'` angezeigt wird, können Sie einfach auf Basis der Kundennummer George Blakes Nummer für dieses Konto in der Tabelle `account` suchen und die Kontonummer dann einsetzen, um die entsprechenden Transaktionen zu ermitteln:

```
SELECT t.txn_id, t.txn_type_cd, t.txn_date, t.amount
FROM account a
  INNER JOIN transaction t ON t.account_id = a.account_id
WHERE a.cust_id = 8 AND a.product_cd = 'CHK';
```

In den nächsten Kapiteln werden alle Konzepte aus diesen Abfragen (und viele andere mehr) noch genauer behandelt, aber ich wollte, dass Sie vorab zumindest schon einmal gesehen haben, wie sich solche Abfragen darstellen.

Die obigen Abfragen enthalten drei verschiedene *Klauseln*: `select`, `from` und `where`. Fast jede Abfrage, die Ihnen jemals unter die Augen kommen wird, enthält mindestens diese drei Klauseln, auch wenn es für besondere Zwecke noch einige andere Klauseln gibt. Die Rolle der drei Klauseln könnte man folgendermaßen darstellen:

```
SELECT /* (WÄHLE) eine oder mehrere Sachen */ ...
FROM /* (AUS) einem oder mehreren Orten */ ...
WHERE /* (WOBEI) eine oder mehrere Bedingungen gelten */ ...
```

Die meisten SQL-Implementierungen behandeln alles, was zwischen /* und */ steht, als Kommentar.

Beim Aufbau einer Abfrage müssen Sie zuerst herausfinden, welche Tabelle(n) Sie benötigen, und diese dann in Ihre `from`-Klausel schreiben. Als Nächstes müssen Sie Ihrer `where`-Klausel Bedingungen hinzufügen, um aus diesen Tabellen die Daten herauszufiltern, die Sie nicht interessieren. Zum Schluss müssen Sie entscheiden, welche Spalten aus den verschiedenen Tabellen abgefragt werden müssen, und diese in Ihre `select`-Klausel einfügen. Das folgende einfache Beispiel zeigt, wie man alle Kunden mit dem Nachnamen »Smith« findet:

```
SELECT cust_id, fname
FROM individual
WHERE lname = 'Smith'
```

Diese Abfrage sucht in der `individual`-Tabelle alle Zeilen, deren `lname`-Spalte den String »Smith« aufweist, und gibt die Werte der Spalten `cust_id` und `fname` aus diesen Zeilen zurück.

Doch vermutlich werden Sie Ihre Datenbank nicht nur abfragen, sondern auch Daten darin einfügen und modifizieren. Das folgende Beispiel zeigt, wie man in die `product`-Tabelle eine neue Zeile einfügt:

```
INSERT INTO product (product_cd, name)
VALUES ('CD', 'Certificate of Depisit')
```

Huch, da habe ich mich bei »Deposit« vertippt. Kein Problem. Das lässt sich mit einer `update`-Anweisung bereinigen:

```
UPDATE product
SET name = 'Certificate of Deposit'
WHERE product_cd = 'CD';
```

Beachten Sie, dass die update-Anweisung auch eine where-Klausel enthält, genau wie die select-Anweisung. Die update-Anweisung muss nämlich die zu modifizierenden Zeilen zunächst identifizieren. In diesem Fall sollen nur diejenigen Zeilen geändert werden, deren product_cd-Spalte den String »CD« enthält. Da die Spalte product_cd der Primärschlüssel der product-Tabelle ist, kann man davon ausgehen, dass die update-Anweisung genau eine Zeile ändert (oder gar keine, wenn der Wert in der Tabelle nicht vorkommt). Immer wenn Sie eine SQL-Datenanweisung ausführen, bekommen Sie von der Datenbank-Engine ein Feedback darüber, wie viele Zeilen von Ihrer Anweisung betroffen waren. Sofern Sie ein interaktives Tool wie das zuvor bereits erwähnte Kommandozeilenprogramm mysql benutzen, meldet es Ihnen, wie viele Zeilen:

- von der select-Anweisung zurückgegeben,
- von der insert-Anweisung eingefügt,
- von der update-Anweisung geändert und
- von der delete-Anweisung gelöscht wurden.

Falls Sie eine prozedurale Sprache mit einem der oben genannten Toolkits benutzen, wird das Toolkit nach der Ausführung der SQL-Datenanweisung einen Aufruf absetzen, um dieses Feedback zu bekommen. Schauen Sie sich diese Informationen genau an, um sicherzustellen, dass Ihre Anweisung nichts Unerwünschtes getan hat (wenn Sie zum Beispiel vergessen, eine where-Klausel in eine delete-Anweisung einzufügen, kann jede einzelne Tabellenzeile gelöscht werden!).

Was ist MySQL?

Relationale Datenbanken werden seit über drei Jahrzehnten kommerziell angeboten. Besonders ausgereifte und populäre kommerzielle Produkte sind:

- Oracle Database von der Oracle Corporation
- SQL Server von Microsoft
- DB2 Universal Database von IBM

Alle diese Datenbankserver tun annähernd das Gleiche, auch wenn einige von ihnen besser gerüstet sind, um mit sehr großen Datenmengen oder sehr hohem Durchsatz fertig zu werden. Wieder andere können besser mit Objekten, großen Dateien oder XML-Dokumenten umgehen. Darüber hinaus sind alle diese Server an den neuesten ANSI-SQL-Standard angepasst worden. Das ist eine gute Sache, und ich werde besonderen Wert darauf legen, Ihnen das Schreiben von SQL-Anweisungen beizubringen, die so gut wie unverändert auf allen diesen Plattformen laufen.

Zusätzlich zu den kommerziellen Anbietern hat die Open-Source-Gemeinde in den letzten zwei Jahrzehnten viel dafür getan, eine gangbare Alternative zu schaffen. Die beiden beliebtesten Open-Source-Datenbankserver sind PostgreSQL und MySQL. Der MySQL-Server ist frei erhältlich und nach meinen Erfahrungen ex-

trem einfach zu laden und zu installieren. Aus diesen Gründen habe ich beschlossen, alle Beispiele in diesem Buch auf meiner MySQL-Datenbank (Version 8.0) auszuführen und das Kommandozeilentool `mysql` zur Formatierung der Ergebnismengen einzusetzen. Selbst wenn Sie bereits einen anderen Server benutzen und nicht auf MySQL umsteigen möchten, bitte ich Sie, den neuesten MySQL-Server zu installieren, das Schema und die Daten der Musterdatenbank zu laden und mit den Daten und Beispielen dieses Buchs zu experimentieren.

Behalten Sie aber dennoch Folgendes im Gedächtnis:

Dies ist kein Buch über die SQL-Implementierung von MySQL.

Nein, dieses Buch soll vermitteln, wie SQL-Anweisungen geschrieben werden, die ohne Änderungen auf MySQL und mit geringfügigen oder auch ohne Änderungen auf neueren Versionen von Oracle Database, DB2 und SQL Server laufen.

SQL unplugged

Die Welt der Datenbanken hat sich ganz schön verändert, seit die letzte Auflage dieses Buchs erschienen ist. Relationale Datenbanken werden zwar immer noch sehr oft eingesetzt, und das wird wohl auch noch eine ganze Weile so bleiben, aber es sind auch neue Datenbanktechnologien entstanden, um die Bedürfnisse von Firmen wie Amazon oder Google zu befriedigen. Zu diesen Technologien gehören Hadoop, Spark, NoSQL und NewSQL, bei denen es sich um verteilte, skalierbare Systeme handelt, die meist auf Clustern aus normalen Servern deployt werden. Es würde zwar den Rahmen dieses Buchs sprengen, diese Techniken detailliert zu behandeln, aber sie haben alle etwas mit relationalen Datenbanken gemeinsam: SQL.

Da Organisationen Daten häufig mithilfe verschiedener Technologien ablegen, gibt es den Bedarf, SQL von einem bestimmten Datenbankserver abzukoppeln und einen Service bereitzustellen, der mehrere Datenbanken abdecken kann. So muss vielleicht ein Report Daten aus Oracle, Hadoop sowie JSON-, CSV- und Unix-Logdateien zusammenführen. Es ist eine neue Generation von Werkzeugen entstanden, die solche Anforderungen bedienen. Eines der vielversprechendsten ist Apache Drill, bei der es sich um eine Open-Source-Query-Engine handelt, die es den Anwendern erlaubt, Abfragen zu schreiben, die auf Daten aus so gut wie allen Datenbanken oder Dateisystemen zugreifen. Wir werden uns Apache Drill in Kapitel 18 anschauen.

Weiteres Vorgehen

Das übergreifende Ziel der nächsten vier Kapitel ist, SQL-Datenanweisungen mit besonderer Berücksichtigung der drei wichtigsten Klauseln der `select`-Anweisung einzuführen. Darüber hinaus werden viele Beispiele gezeigt, die das Sakila-Schema nutzen, das im folgenden Kapitel eingeführt und für alle Beispiele dieses Buchs ver-

wendet wird. Ich hoffe, dass Sie durch die Vertrautheit mit einer einzigen Datenbank schneller zum Kern eines Beispiels vorstoßen, ohne jedes Mal innehalten und nachschauen zu müssen, welche Tabellen denn dieses Mal wieder benötigt werden. Sollte es Ihnen zu langweilig werden, immer mit der gleichen Menge an Tabellen zu arbeiten, können Sie die Beispieldatenbank gern mit zusätzlichen Tabellen ausstatten oder sich zum Experimentieren Ihre eigene Datenbank ausdenken.

Nachdem Sie ein solides Grundlagenwissen aufgebaut haben, gehen die nachfolgenden Kapitel mehr in die Tiefe und stellen zusätzliche Konzepte vor, die zumeist voneinander unabhängig sind. Falls Sie erst mal verwirrt sind, blättern Sie einfach weiter und kommen später auf das noch nicht gelesene Kapitel zurück. Wenn Sie das Buch beendet und alle Beispiele durchgearbeitet haben, sind Sie bereits auf dem besten Wege, ein mit allen Wassern gewaschener SQL-Guru zu werden.

Wer noch mehr über relationale Datenbanken, die Geschichte der computergestützten Datenbanksysteme oder die Sprache SQL erfahren möchte, sollte sich folgende Quellen anschauen:

- *Database in Depth: Relational Theory for Practitioners* von C. J. Date (O'Reilly)
- *An Introduction to Database Systems*, 8. Auflage, von C. J. Date (Addison-Wesley)
- *The Database Relational Model: A Retrospective Review and Analysis* von C. J. Date (Addison-Wesley)
- Wikipedia-Unterartikel zur Definition von »Database Management System« (*https://oreil.ly/sj2xR*)

KAPITEL 2

Datenbanken erstellen und mit Daten füllen

In diesem Kapitel erfahren Sie alles Notwendige, um Ihre erste eigene Datenbank mit den Tabellen und Daten für die Beispiele in diesem Buch anzulegen. Außerdem lernen Sie die verschiedenen Datentypen kennen und sehen, wie man die passenden Tabellen erzeugt, um mit ihnen arbeiten zu können. Da die Beispiele dieses Buchs in einer MySQL-Datenbank ausgeführt werden, ist dieses Kapitel von den Features und der Syntax von MySQL beeinflusst, aber das meiste ist auch für alle anderen Server anwendbar.

Eine MySQL-Datenbank anlegen

Um mit den Daten zu experimentieren, die für die Beispiele in diesem Buch genutzt wurden, haben Sie zwei Möglichkeiten:

- Laden Sie die MySQL-Server-Version 8.0 (oder neuer) herunter und installieren Sie sie. Holen Sie sich dann die Sakila-Beispieldatenbank, die Sie unter *https://dev.mysql.com/doc/index-other.html* finden.
- Rufen Sie *https://www.katacoda.com/mysql-db-sandbox/scenarios/mysql-sandbox* auf, um auf die MySQL-Sandbox zuzugreifen, in der die Sakila-Beispieldatenbank in einer MySQL-Instanz geladen ist. Sie müssen dafür einen (kostenlosen) Katacoda-Account einrichten. Dann klicken Sie auf den Button *Start Scenario*.

Haben Sie die zweite Möglichkeit gewählt, wird nach dem Starten des Szenarios ein MySQL-Server installiert und gestartet, und anschließend wird das Sakila-Schema mit seinen Daten geladen. Ist alles erledigt, erscheint ein Standardprompt `mysql>`, und Sie können damit beginnen, die Beispieldatenbank abzufragen. Das ist die einfachste Option, und ich vermute, die meisten Leser werden sie wählen – wenn das für Sie gut genug klingt, können Sie einfach zum nächsten Abschnitt springen.

Möchten Sie lieber Ihre eigene Kopie der Daten nutzen und Änderungen daran dauerhaft machen oder sind Sie einfach daran interessiert, den MySQL-Server auf Ihrem eigenen Rechner zu installieren, bevorzugen Sie vielleicht die erste Option. Sie können sich auch dazu entscheiden, einen MySQL-Server zu nutzen, der in der

Cloud gehostet wird, wie zum Beispiel bei Amazon Web Services oder Google Cloud. Sie müssen dann auf jeden Fall die Installation und die Konfiguration selbst vornehmen, was aber nicht Bestandteil dieses Buchs ist. Ist Ihre Datenbank bereit, werden Sie noch ein paar weitere Schritte ausführen müssen, um die Sakila-Beispieldatenbank zu laden.

Als Erstes müssen Sie das `mysql`-Kommandozeilentool starten und ein Passwort angeben, danach führen Sie die folgenden Schritte aus:

1. Öffnen Sie *https://dev.mysql.com/doc/index-other.html* und laden Sie die Dateien für *sakila database* aus dem Abschnitt *Example Databases* herunter.
2. Legen Sie sie in einem lokalen Verzeichnis wie *C:\temp\sakila-db* ab (das nutzen wir für die nächsten beiden Schritte, aber Sie können natürlich gern einen anderen Pfad wählen).
3. Geben Sie `source c:\temp\sakila-db\sakila-schema.sql;` ein und drücken Sie die Taste Enter.
4. Geben Sie `source c:\temp\sakila-db\sakila-data.sql;` ein und drücken Sie die Taste Enter.

Jetzt sollten Sie eine laufende Datenbank haben, die mit all den Daten gefüllt ist, die Sie für die Beispiele in diesem Buch benötigen.

Die Sakila-Beispieldatenbank wird von MySQL bereitgestellt, sie ist über die New BSD License lizenziert. Sakila enthält Daten für eine fiktive Videothek mit Tabellen wie `store`, `inventory`, `film`, `customer` oder `payment`. Während die Zeit echter Videotheken eher vorbei ist, können wir sie mit ein wenig Fantasie zu einer Streaming-Firma machen, wenn wir die Tabellen `staff` und `address` ignorieren und `store` in `streaming_service` umbenennen. Aber für die Beispiele hier im Buch bleibe ich bei der ursprünglichen Version.

Das mysql-Kommandozeilentool

Sofern Sie keine temporäre Datenbanksession nutzen (die zweite Variante im vorigen Abschnitt), müssen Sie das Kommandozeilentool `mysql` aufrufen, um mit der Datenbank interagieren zu können. Dazu öffnen Sie eine Windows- oder Unix-Shell und führen das Tool `mysql` aus. Melden Sie sich beispielsweise mit dem `root`-Account an, könnten Sie das wie folgt tun:

```
mysql -u root -p;
```

Sie werden nach Ihrem Passwort gefragt, danach sehen Sie den `mysql>`-Prompt. Um sich alle verfügbaren Datenbanken anzeigen zu lassen, können Sie den folgenden Befehl einsetzen:

```
mysql> show databases;

+--------------------+
| Database           |
```

```
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
+--------------------+
5 rows in set (0.01 sec)
```

Da Sie die Sakila-Datenbank verwenden werden, müssen Sie diese mit der `use`-Anweisung spezifizieren:

```
mysql> use sakila;
Database changed
```

Immer wenn Sie das `mysql`-Kommandozeilentool aufrufen, können Sie den Benutzernamen und die zu verwendende Datenbank wie folgt angeben:

```
mysql -u lrngsql -p sakila;
```

Das erspart Ihnen, jedes Mal, wenn Sie das Tool starten, `use sakila;` einzugeben. Sie haben jetzt eine Session erzeugt und die Datenbank festgelegt – nun können Sie SQL-Anweisungen ausführen und die Ergebnisse anschauen. Wenn Sie zum Beispiel das Datum und die Uhrzeit wissen möchten, können Sie folgende Abfrage durchführen:

```
mysql> SELECT now();
+---------------------+
| now()               |
+---------------------+
| 2019-04-04 20:44:26 |
+---------------------+
1 row in set (0.01 sec)
```

Die `now()`-Funktion ist eine eingebaute MySQL-Funktion, die das Datum und die aktuelle Uhrzeit zurückliefert. Wie Sie sehen, formatiert das `mysql`-Kommandozeilentool die Ergebnisse Ihrer Abfragen in einem rechteckigen Kasten, der durch die Zeichen +, – und | abgegrenzt wird. Wenn alle Ergebnisse angezeigt wurden (in diesem Fall nur eine einzige Zeile), zeigt das `mysql`-Kommandozeilentool außerdem, wie viele Zeilen zurückgegeben wurden und wie lange die Ausführung der SQL-Anweisung dauerte.

Fehlende from-Klauseln

In manchen Datenbankservern ist es gar nicht möglich, eine Abfrage ohne eine `from`-Klausel abzusetzen, in der nicht mindestens eine Tabelle genannt wird. Die Oracle Database ist beispielsweise ein viel genutzter Server, für den dieses gilt.

Für Fälle, in denen lediglich eine Funktion aufgerufen werden muss, bietet Oracle eine Tabelle namens `dual` mit einer einzigen Spalte namens `dummy`, die nur eine einzige Datenzeile enthält. Um mit Oracle Database kompatibel zu sein, stellt MySQL ebenfalls eine `dual`-Tabelle zur Verfügung. Die obige Abfrage des Datums und der Uhrzeit könnte man also auch folgendermaßen schreiben:

```
mysql> SELECT now()
       FROM dual;
+---------------------+
| now()               |
+---------------------+
| 2009-05-06 16:48:46 |
+---------------------+
1 row in set (0.01 sec)
```

Wenn Sie Oracle nicht benutzen und auch keine Notwendigkeit darin sehen, mit Oracle Kompatibilität herzustellen, können Sie die `dual`-Tabelle vergessen.

Sobald Sie mit dem `mysql`-Kommandozeilentool fertig sind, geben Sie einfach `quit;` oder `exit;` ein, um zur Unix- oder Windows-Eingabeaufforderung zurückzukehren.

MySQL-Datentypen

Generell haben alle populären Datenbankserver die Kapazität, die gleichen Datentypen zu speichern, etwa Strings, Datumswerte und Zahlen. Unterschiede gibt es typischerweise bei den spezielleren Datentypen, beispielsweise in XML- oder JSON-Dokumenten oder auch Geodaten. Da dieses Buch eine Einführung in SQL ist und 98 % aller Spalten, mit denen Sie es zu tun haben werden, einfache Datentypen sind, befasst sich dieses Kapitel nur mit den Datentypen für Zeichen, Datumswerte (oder temporale Werte) und Zahlen. Der Einsatz von SQL zum Abfragen von JSON-Dokumenten wird in Kapitel 18 behandelt.

Zeichendaten

Zeichendaten können als Strings mit fester oder variabler Länge gespeichert werden. Der Unterschied besteht darin, dass Strings fester Länge nach rechts mit Leerzeichen aufgefüllt werden und immer die gleiche Anzahl von Bytes benötigen. Strings mit variabler Länge werden hingegen nicht mit Leerzeichen aufgefüllt und benötigen nicht immer die gleiche Anzahl Bytes. Wenn Sie eine Zeichenspalte definieren, müssen Sie angeben, wie lang ein String, der in dieser Spalte gespeichert wird, maximal sein darf. Möchten Sie zum Beispiel Strings mit höchstens 20 Zeichen speichern, könnten Sie folgende Definitionen verwenden:

```
char(20)    /* feste Länge */
varchar(20) /* variable Länge */
```