GANZ EINFACH

Datenbanken

Das umfassende Handbuch

Datenbanken verstehen, bauen und skalieren



Danilo Sieren

Danilo Sieren

Datenbanken

Daten modellieren und mit SQL auswerten
Datenbanken verlässlich betreiben

Überarbeitete und erweiterte
Neuauflage 2026

Verstehen · Anwenden · Überprüfen

Vorwort

Was du mit diesem Buch erreichen kannst

Eine Datenbank hilft dir, zusammengehörige Informationen verlässlich zu speichern und gezielt auszuwerten. Du brauchst dafür weder ein großes Unternehmen noch einen leistungsstarken Server. Bereits eine private Büchersammlung wirft dieselben Grundfragen auf wie ein Warenwirtschaftssystem. Was ist ein einzelner Datensatz? Welche Angaben müssen eindeutig sein? Welche Änderungen gehören zusammen? Und wie lässt sich nachweisen, dass eine Sicherung wirklich wiederherstellbar ist?

Dieses Buch führt dich von diesen Fragen zu einer praktischen Arbeitsweise. Du baust eine kleine Bibliothek auf, formulierst SQL-Abfragen und entwickelst anschließend einen vereinfachten Bioladen. An diesen beiden Beispielen lernst du Datenmodelle, Schlüssel, Beziehungen, Auswertungen und Transaktionen kennen. Später unterscheidest du lokale Datenbanken von Serverdiensten und lernst einzuschätzen, wann Replikation, Cloudangebote, Dokumentdatenbanken oder eine analytische Datenbank einen begründeten Nutzen haben.

Der Schwerpunkt liegt auf verständlichen Entscheidungen und überprüfbaren Ergebnissen. Ein korrektes Ergebnis mit fünf Zeilen ist zum Lernen wertvoller als eine beeindruckende Architektur, deren Fehler niemand bemerkt. Die fortgeschrittenen Kapitel knüpfen deshalb immer an die Fragen der kleinen Projekte an. Ein Ereignisstrom muss ebenfalls Duplikate behandeln. Eine Vektorsuche muss ebenfalls Zugriffsrechte beachten. Ein verteilter Dienst braucht ebenfalls einen Wiederherstellungsweg.

Wie du die Übungen bearbeitest

Die ersten SQL-Übungen verwenden SQLite. Du arbeitest in einer eigenen Übungsdatei auf einem lokalen Datenträger. Für den Einstieg genügt DB Browser for SQLite oder die SQLite-Kommandozeile. Ein Server, ein kostenpflichtiges Konto und echte Kundeninformationen sind dafür nicht nötig. Die Übungen nennen jeweils das benötigte Ausgangsmodell. SQL für PostgreSQL und andere Systeme ist ausdrücklich gekennzeichnet und gehört nicht ungeprüft in dieselbe SQLite-Datei.

Arbeite die Bibliothekskapitel zunächst der Reihe nach durch. Führe einen vollständigen Befehlsblock aus und vergleiche das Ergebnis mit der Erklärung. Ein erwarteter Fehler ist Teil der Übung, wenn der Text dies ankündigt. Stoppe dann und prüfe die genannte Regel. Kopiere bei einem unerwarteten Fehler nicht einfach weitere Befehle hinterher. Eine fehlende Tabelle am Anfang kann sonst viele Folgefehler erzeugen, die mit dem eigentlichen Lernziel wenig zu tun haben.

Die Programme in diesem Buch verwenden bewusst kleine Datenbestände. Sie zeigen eine vollständige fachliche Aufgabe, ersetzen aber kein Warenwirtschaftssystem und keine geprüfte Kassenlösung. Das Bioladenprojekt enthält beispielsweise keine Zahlungsabwicklung und keine vollständige steuerliche Behandlung. Gerade diese Abgrenzung hilft dir, das Datenmodell fachlich zu beurteilen, statt aus einigen funktionierenden Tabellen voreilig auf einen einsatzfertigen Geschäftsbetrieb zu schließen.

Welche Kenntnisse du mitbringen solltest

Du solltest Dateien und Ordner anlegen, Programme installieren und Text kopieren können. SQL und Datenmodellierung werden von Grund auf erklärt. Für die Python-Übungen reicht zunächst, eine Textdatei mit der Endung .py zu speichern und ein Programm mit Python 3 zu starten. Die Beispiele verwenden die

mit Python gelieferte SQLite-Schnittstelle; bei ungewöhnlichen Python-Distributionen kann dieses optionale Modul fehlen. [D19]

Die Kapitel über Serverbetrieb, Streaming und maschinelles Lernen gehen weiter. Lies sie beim ersten Durchgang als Entscheidungshilfe. Du musst nicht jede dort beschriebene Technik installieren. Wenn deine Aufgabe mit einer lokalen SQLite-Datei gut funktioniert, ist das eine angemessene Lösung. Ein zusätzlicher Dienst ist erst dann ein Fortschritt, wenn er ein konkretes Problem löst und du seine Verwaltung übernehmen kannst.

Inhalt

Vorwort 2
1 Datenbanken an einer Aufgabe verstehen 7
2 Die lokale Lernumgebung einrichten 12
3 Tabellen und verlässliche Werte entwerfen 17
4 Mit SELECT präzise Fragen stellen 22
5 Beziehungen mit Fremdschlüsseln abbilden 27
6 Tabellen verbinden und Ergebnisse erklären 33
7 Daten kontrolliert ändern 38
8 Summen und Berichte richtig berechnen 43
9 Modelle normalisieren und Geschichte bewahren 48
10 Den Bioladen als Lernprojekt modellieren 53
11 Transaktionen und gleichzeitige Zugriffe beherrschen 58
12 Python sicher mit der Datenbank verbinden 63
13 Daten importieren und Qualität sichern 69
14 Abfragepläne und Indizes gezielt einsetzen 74
15 Schemaänderungen und Datenbankpflege planen 78
16 Vom lokalen Bestand zum Datenbankserver wechseln 84
17 Rollen Verbindungen und Geheimnisse schützen 89
18 Sicherungen erstellen und Wiederherstellung prüfen 93
19 Replikation und Hochverfügbarkeit nüchtern planen 98
20 Cloudangebote nach Verantwortung auswählen 102
21 APIs und Anwendungen verlässlich anbinden 106
22 JSON und Dokumentmodelle bewusst verwenden 110
23 Cache Graphen und Zeitreihen einordnen 114
24 Analytische Datenbestände aufbauen 118
25 Ereignisströme und Wiederholungen verstehen 122

26 Partitionierung und Sharding begründet einsetzen 127
27 Maschinelles Lernen und Vektorsuche beurteilen 131
28 Datenschutz in das Modell einbauen 135
29 Fehler und Leistung im Betrieb beobachten 139
30 Eine tragfähige Datenarchitektur entscheiden 143
Anhang A Die Bibliothek vollständig aufbauen 148
Anhang B Einen Verkauf robust programmieren 154
Anhang C CSV Daten prüfen und übernehmen 160
Anhang D Typische Fehler systematisch untersuchen 165
Anhang E Vorlagen für Entwurf und Betrieb 172
Anhang F Begriffe zum Nachschlagen 177
Quellen und offizielle Dokumentation 181
Ausführliches Inhaltsverzeichnis 185
Stichwortverzeichnis 193
Impressum 197

1 Datenbanken an einer Aufgabe verstehen

1.1 Mit einer konkreten Frage beginnen

Du möchtest wissen, wem du ein Buch geliehen hast. In einer Notiz steht der Titel, in einer Nachricht der Name und im Kalender vielleicht ein Rückgabetermin. Die Informationen sind vorhanden, lassen sich aber nur mühsam gemeinsam betrachten. Eine Datenbank ordnet sie so, dass du eine präzise Frage stellen kannst. Zum Beispiel suchst du alle noch offenen Ausleihen mit einem Rückgabetermin vor dem heutigen Tag.

Die Frage legt bereits fest, welche Informationen du brauchst. Du benötigst ein erkennbares Exemplar, eine ausleihende Person, den Beginn der Ausleihe, die vereinbarte Frist und gegebenenfalls den tatsächlichen Rückgabetag. Ein schönes Titelfeld allein reicht dafür nicht. Wenn du zwei Exemplare desselben Romans besitzt, muss außerdem feststehen, welches Exemplar unterwegs ist. Die fachliche Bedeutung der Daten geht dem technischen Tabellenentwurf voraus.

Notiere deshalb vor dem ersten Programmstart drei Fragen, die deine Sammlung später beantworten soll. Prüfe bei jeder Frage, ob die dafür nötigen Angaben überhaupt entstehen. Ein System kann die Rückgabe nicht zuverlässig feststellen, wenn niemand sie erfasst. Auch die beste Abfragesprache kann eine fehlende Beobachtung nicht nachträglich herbeizaubern.

1.2 Datenbank und Verwaltungsprogramm unterscheiden

Die Datenbank enthält den geordneten Datenbestand. Das Datenbankmanagementsystem verwaltet diesen Bestand und setzt Regeln durch. Eine Anwendung bietet die Oberfläche für die

eigentliche Arbeit. Bei SQLite ist die Datenbank häufig eine einzelne Hauptdatei; die Datenbanksoftware läuft als Bibliothek im verwendeten Programm. PostgreSQL arbeitet dagegen als eigener Serverprozess, zu dem sich Anwendungen verbinden. Diese Unterscheidung erklärt viele spätere Unterschiede bei Benutzerrechten und Sicherungen.

DB Browser for SQLite ist ein grafisches Verwaltungsprogramm. Es zeigt Tabellen und führt SQL aus, ist aber nicht selbst ein Ersatz für die SQLite-Datenbankengine. Ebenso ist eine Tabellenansicht in einem Serverwerkzeug nur eine Darstellung. Wenn du dort eine Zeile änderst, kann im Hintergrund eine Transaktion mit Prüfungen stattfinden. Beurteile eine Datenbank daher nicht allein nach dem sichtbaren Raster.

Eine CSV-Datei kann viele Zeilen enthalten, besitzt jedoch von sich aus keine Fremdschlüsselprüfung und keine Transaktionsverwaltung für mehrere verknüpfte Tabellen. Eine Tabellenkalkulation bietet andere Stärken, etwa freie Berechnungen und anschauliche Planung. Für einen einmaligen Haushaltsplan kann sie gut passen. Wenn mehrere Datenbestände konsistent miteinander arbeiten sollen, werden explizite Datenbankregeln besonders nützlich.

1.3 Einen Datensatz fachlich abgrenzen

Ein Datensatz beschreibt eine Sache oder ein Ereignis auf einer festgelegten Ebene. In einer Büchertabelle könnte eine Zeile einen veröffentlichten Titel beschreiben. In einer Exemplartabelle beschreibt sie dagegen ein bestimmtes physisches Buch. Diese Ebenen sind nicht austauschbar. Ein Titel kann mehrere Exemplare besitzen; jedes Exemplar hat seinen eigenen Standort und seine eigene Ausleihgeschichte.

Für die ersten Übungen vereinfachen wir das Modell. Jede Buchzeile steht für genau ein ausleihbares Exemplar. Der Titel darf mehrfach vorkommen. Später erweitern wir das Modell um

Autorinnen, Autoren und weitere Beziehungen. Du erkennst dadurch, welche Entscheidung eine Lernvereinfachung ist und welche Regel auch bei einem größeren Bestand benötigt wird.

Im Bioladen unterscheiden wir entsprechend das Produkt vom Verkauf und die Verkaufsposition vom ganzen Bon. Ein Bon kann mehrere Positionen enthalten. Ein Produkt kann in vielen Positionen vorkommen. Wenn du diese Ebenen in eine einzige Zeile presst, entstehen schnell Wiederholungen oder unklare Summen. Eine brauchbare Struktur macht deutlich, worüber eine Zeile eine Aussage trifft.

1.4 Korrektheit als überprüfbare Eigenschaft behandeln

Eine Datenbank kennt zunächst die von dir festgelegten Regeln. Sie erkennt beispielsweise einen negativen Preis, wenn du dafür eine passende Bedingung definierst. Sie erkennt nicht automatisch, dass 999 Euro für ein bestimmtes Taschenbuch wahrscheinlich ein Tippfehler sind. Fachliche Plausibilität erfordert zusätzliche Regeln und oft eine Prüfung durch Menschen.

Unterscheide technische Integrität und fachliche Richtigkeit. Ein Fremdschlüssel kann sicherstellen, dass eine Ausleihe auf eine vorhandene Person verweist. Er beweist nicht, dass diese Person das Buch tatsächlich erhalten hat. Ein korrekt gespeicherter Zeitstempel beweist nicht, dass die Uhr richtig eingestellt war. Diese Grenzen werden später bei Protokollen, Messwerten und KI-Auswertungen erneut wichtig.

Definiere für jede wichtige Aufgabe ein sichtbares Prüfergebnis. Nach einer Rückgabe darf die Ausleihe nicht mehr in der Liste offener Vorgänge erscheinen. Nach einem Verkauf muss der erwartete Bestand vorliegen. Nach einer Wiederherstellung müssen nicht nur Tabellen existieren, sondern auch einige bekannte Vorgänge richtig gelesen werden können. So wächst Vertrauen aus Beobachtungen.

1.5 Den Umfang bewusst begrenzen

Für den Einstieg führen wir keine echten personenbezogenen Daten. Namen wie Mira und Paul bezeichnen erfundene Personen. Die Bibliothek erhält wenige Bücher; der Bioladen verkauft nur ganze Stückzahlen und verwendet Eurocent. Diese Begrenzungen halten die Beispiele überschaubar. Gewichtsware, Pfand, mehrere Währungen oder gemeinsame Konten sind jeweils eigene Modellierungsaufgaben.

Schreibe solche Grenzen auf. Eine stillschweigende Annahme wird sonst später zur Fehlerquelle. Wenn eine Menge immer ganzzahlig sein soll, muss die Anwendung das prüfen und das Schema diese Absicht möglichst ausdrücken. Wenn ein Preis nur in einer Währung gilt, darf die Spalte nicht plötzlich Werte unterschiedlicher Währungen mischen.

Auch der Betrieb gehört zum Umfang. Eine Datei auf deinem Rechner kann für eine Einzelperson genügen. Mehrere gleichzeitige Kassen an verschiedenen Standorten stellen andere Anforderungen. Es wäre unzweckmäßig, bereits für die ersten fünf Bücher einen verteilten Cluster aufzubauen. Ebenso wäre es unvorsichtig, eine lokale Lernlösung ungeprüft als Mehrbenutzersystem zu verwenden.

1.6 Werkstatt Deinen Datenauftrag formulieren

Schreibe auf einer halben Seite auf, welche Sammlung du selbst verwalten möchtest. Beschreibe zuerst die Hauptfrage in einem Satz. Ergänze anschließend drei typische Vorgänge und ein Beispiel für einen unerwünschten Zustand. Eine Filmsammlung könnte Regiearbeiten zuordnen, Sichtungen erfassen und eine Merkliste führen. Ein unerwünschter Zustand wäre eine Sichtung, die auf einen nicht vorhandenen Film verweist.

Bestimme danach die kleinste sinnvolle Zeileneinheit. Vermeide dabei Wörter wie alles oder Informationen, die keine klare Abgrenzung geben. Schreibe stattdessen beispielsweise: Eine Zeile beschreibt eine bestimmte Vorführung an einem bestimmten Tag. Prüfe schließlich, welche Angabe du später ändern können musst und welche Angabe einen historischen Vorgang unverändert beschreiben soll.

Bewahre diesen Auftrag auf. Du wirst ihn bei Schlüsseln, Beziehungen und Sicherungen wieder verwenden. Ein Datenmodell ist dann gelungen, wenn es deine vereinbarten Vorgänge verständlich abbildet und Fehler begrenzt. Die Zahl der Tabellen ist dafür kein eigenständiges Qualitätsmerkmal.

2 Die lokale Lernumgebung einrichten

2.1 SQLite und ein Werkzeug auswählen

Für die Übungen benötigst du SQLite ab Version 3.37, weil wir strenge Tabellen verwenden. Die Versionsnummer des Verwaltungsprogramms ist davon zu unterscheiden. Ein grafisches Werkzeug kann eine eigene SQLite-Bibliothek mitbringen; das Terminalprogramm und Python können auf demselben Rechner andere Bibliotheksversionen verwenden. Prüfe deshalb die tatsächlich verwendete Engine mit einer SQL-Abfrage. [D04]

```
SELECT sqlite_version();
```

DB Browser for SQLite ist für Windows, macOS und Linux erhältlich. Lade es über die Projektseite und wähle das Paket für dein Betriebssystem und gegebenenfalls deine Prozessorarchitektur. Die Projektseite führt auch zur Dokumentation. Die Übungen setzen keine bestimmte Sprache der Oberfläche voraus; SQL-Schlüsselwörter und Tabellennamen bleiben in allen Sprachfassungen gleich. [D02]

Alternativ nutzt du die SQLite-Kommandozeile. Unter macOS und manchen Linux-Systemen ist ein sqlite3-Programm bereits vorhanden. Unter Windows kannst du die offiziellen Kommandozeilenwerkzeuge entpacken und aus ihrem Ordner starten. Prüfe die Version auch hier ausdrücklich. Die mit dem Betriebssystem gelieferte Fassung muss nicht mit der neuesten Projektversion übereinstimmen.

2.2 Einen eigenen Arbeitsordner anlegen

Lege einen Ordner Datenbankkurs auf einem lokalen Datenträger an. Darin entstehen bibliothek.db, spaeter

bioladen.db sowie deine SQL-Dateien. Halte Lernbestände von vorhandenen Anwendungsdatenbanken getrennt. Eine Datei, die zu einer Fotoverwaltung oder einem Passwortmanager gehört, ist kein geeigneter Platz für eigene Experimente.

Verwende für die aktiven Übungsdateien keinen gleichzeitig synchronisierten Cloudordner und keine Netzwerkfreigabe. Eine Synchronisation ersetzt weder Dateisperren noch eine datenbankgerechte Sicherung. SQLite kann neben der Hauptdatei weitere Dateien verwalten; insbesondere im WAL-Modus gehören noch nicht übertragene Änderungen möglicherweise zur WAL-Datei. Das isolierte Kopieren der Hauptdatei kann dann einen unvollständigen Stand liefern. [D12]

Ein SQL-Skript ist eine Textdatei mit Befehlen. Die Endung .sql hilft dir, es von einer Datenbankdatei zu unterscheiden. Das Skript enthält nicht automatisch den kompletten aktuellen Datenbestand. Speichere daher sowohl deine nachvollziehbaren Lernschritte als auch bewusst erstellte Sicherungen. Verwechsle die beiden Arten von Dateien nicht.

2.3 Die Datenbank im grafischen Werkzeug öffnen

Erstelle in DB Browser eine neue Datenbankdatei über die Funktion für eine neue Datenbank und speichere sie als bibliothek.db. Wenn das Werkzeug sofort einen Tabellenentwurf anbietet, kannst du diesen Dialog abbrechen; die Tabellen entstehen später mit dem abgedruckten SQL. Öffne anschließend den Bereich zum Ausführen von SQL, in einer englischen Oberfläche Execute SQL.

Führe die Versionsabfrage aus Abschnitt 2.1 aus. Ein Ergebnisraster mit einer Versionsnummer zeigt, dass die Verbindung arbeitet. Achte darauf, ob das Werkzeug alle Befehle, den markierten Ausschnitt oder nur die aktuelle Anweisung

ausführt. Wenn du nur einen Teil eines längeren Skripts markierst, können später erwartete Tabellen fehlen.

DB Browser unterscheidet ausstehende Änderungen und ausdrücklich geschriebene Änderungen. Prüfe den Status der Verbindung und schließe eine begonnene Transaktion bewusst ab. Eine zweite Anwendung sieht unbestätigte Änderungen nicht zuverlässig so wie die erste. Wenn du eine Datei nach einer Änderung in einem anderen Werkzeug öffnest, ist ein abweichender Stand deshalb noch kein Beweis für Datenverlust.

2.4 Die Kommandozeile als Alternative nutzen

Öffne ein Terminal in deinem Arbeitsordner. Mit dem folgenden Aufruf öffnest du bibliothek.db. Existiert die Datei noch nicht, kann SQLite sie bei Bedarf anlegen. Ein Tippfehler im Dateinamen kann deshalb zu einer unerwartet leeren zweiten Datenbank führen. Kontrolliere den Pfad, wenn plötzlich Tabellen fehlen.

```
sqlite3 bibliothek.db
```

Innerhalb der SQLite-Shell gelten SQL-Anweisungen und besondere Punktbefehle. SQL endet gewöhnlich mit einem Semikolon. Ein Punktbefehl steht dagegen auf einer eigenen Zeile und erhält kein abschließendes SQL-Semikolon. Die folgende Folge verbessert die Anzeige und zeigt die geöffneten Datenbanken. Sie gehört in die Shell, nicht in das SQL-Feld eines beliebigen grafischen Werkzeugs. [D01]

```
.headers on
.mode column
.databases
SELECT sqlite_version();
```

Mit .quit beendest du die Shell. Ein geöffneter Editor oder ein Terminalfenster ist nicht automatisch dieselbe Datenbankverbindung. Wenn du zwischen Programmen

wechselst, prüfe Pfad und Verbindungszustand neu. Besonders bei Übungen mit mehreren Verbindungen ist diese Zuordnung wichtig.

2.5 Die Fremdschlüsselprüfung einschalten

SQLite muss Fremdschlüssel für jede Verbindung wirksam prüfen. Setze die folgende Einstellung direkt nach dem Öffnen, bevor du eine Transaktion beginnst. Die anschließende Abfrage soll 1 zurückgeben. Eine gesetzte Definition im Tabellenschema allein genügt nicht als Nachweis, dass die Verbindung sie gerade durchsetzt. [D05]

```
PRAGMA foreign_keys = ON;
PRAGMA foreign_keys;
```

Innerhalb einer bereits laufenden Mehranweisungstransaktion bewirkt ein Umschalten dieser Einstellung nicht die gewünschte Änderung. Falls das Ergebnis 0 bleibt, beende den aktuellen Versuch kontrolliert und öffne eine neue Verbindung. Schreibe diese Zeilen später auch in Python-Verbindungen vor den Beginn der eigentlichen Arbeit.

Ändere zu Beginn keine weiteren PRAGMA-Einstellungen auf Verdacht. Ein im Internet gefundener schneller Modus kann die Dauerhaftigkeit beeinflussen. Für die ersten kleinen Datenbestände sind nachvollziehbare Regeln wichtiger als ein kaum messbarer Geschwindigkeitsgewinn. Einstellungen, deren Zweck du nicht erklären kannst, gehören noch nicht in dein Startskript.

2.6 Werkstatt Den Ausgangszustand protokollieren

Notiere Werkzeug, Betriebssystem, SQLite-Version und den vollständigen Pfad deiner Übungsdatei. Speichere die Versionsabfrage und die Fremdschlüsseleinstellung als start.sql. Führe beide erneut aus, nachdem du das Werkzeug geschlossen und wieder geöffnet hast. Damit übst du bereits, einen Zustand reproduzierbar herzustellen.

Prüfe dann einen absichtlichen Tippfehler nur gedanklich: Würdest du erkennen, dass bibliothekk.db eine andere Datei als bibliothek.db ist? Suche im Werkzeug die Anzeige des aktuellen Dateipfads. Diese unscheinbare Kontrolle hilft später oft mehr als ein komplizierter Reparaturversuch.

Dein Ergebnis ist eine erreichbare lokale Datenbank mit bekannter Engineversion und aktiver Fremdschlüsselprüfung. Es sind noch keine Fachdatensätze nötig. Erst wenn dieser Ausgangspunkt klar ist, beginnst du mit dem Schema. Eine saubere Lernumgebung spart dir das Rätselraten, ob ein Fehler vom Modell, vom Befehl oder vom falschen Ziel stammt.

3 Tabellen und verlässliche Werte entwerfen

3.1 Eine Zeile eindeutig wiederfinden

Unsere erste Tabelle heißt buch. Jede Zeile steht für ein bestimmtes ausleihbares Exemplar. Die Kennung id ist der Primärschlüssel. Ein Titel ist dafür ungeeignet, weil mehrere Exemplare denselben Titel tragen können und ein Schreibfehler korrigierbar bleiben soll. Eine stabile Kennung trennt die Identität der Zeile von ihrer sichtbaren Beschreibung.

Für SQLite verwenden wir INTEGER PRIMARY KEY. Genau diese Typbezeichnung besitzt eine besondere Bedeutung: Sie kann die interne Zeilenkennung vertreten. AUTOINCREMENT ist dafür normalerweise nicht erforderlich und löst ein anderes Problem, nämlich die Wiederverwendung zuvor vergebener Kennungen. Fortlaufende lückenlose Nummern darfst du daraus nicht ableiten. Eine technische ID ist auch keine Rechnungsnummer. [D06]

Gib dem Primärschlüssel keine Bedeutung, die er nicht besitzt. Eine höhere ID beweist beispielsweise nicht allgemein, dass ein Ereignis später stattgefunden hat. Importierte Daten können Kennungen vorgeben; Transaktionen können scheitern. Für zeitliche Aussagen brauchst du ein eigenes fachliches Datum oder einen passenden Zeitstempel.

3.2 Text und Zahlen nach ihrer Bedeutung wählen

Ein Buchtitel ist Text. Ein Erscheinungsjahr kann eine ganze Zahl sein, solange dein Modell nur gewöhnliche Jahresangaben benötigt. Eine Postleitzahl gehört dagegen gewöhnlich in ein Textfeld. Führende Nullen sind dort relevant; außerdem ist eine

Postleitzahl kein Wert, mit dem du sinnvoll addierst. Nicht jedes Feld, das nur Ziffern zeigt, sollte eine Zahl sein.

SQLite kennt im strengen Tabellenmodus unter anderem INTEGER, REAL, TEXT und BLOB. Andere vertraute Typnamen wie DATE oder DECIMAL sind dort nicht zulässig. Für einen Geldbetrag im Lernmodell verwenden wir ganze Eurocent. Ein Kaufpreis von 12,90 Euro wird als 1290 gespeichert. Das vermeidet binäre Rundungsfehler bei den einfachen Ganzzahlrechnungen dieses Projekts. [D03, D04]

Diese Entscheidung löst nicht jede finanzielle Aufgabe. Prozentuale Rabatte, Steueranteile, Wechselkurse und gewichtsbasiert verkaufte Mengen brauchen ausdrücklich vereinbarte Rundungen. PostgreSQL bietet dafür beispielsweise exakte numerische Typen. Auch mit einem exakten Typ musst du festlegen, an welcher Stelle gerundet wird. Ein Datentyp ersetzt keine fachliche Rechenregel.

3.3 Die erste Tabelle erstellen

Führe den folgenden Block in deiner frischen bibliothek.db aus. Die drei Angaben nach der Kennung sind Pflichtfelder, sofern nicht ein Standardwert oder eine spätere Erweiterung etwas anderes vorsieht. Der Preis darf nicht negativ sein. Ein leerer Titel, der nur aus gewöhnlichen Leerzeichen besteht, wird ebenfalls zurückgewiesen.

```
CREATE TABLE buch (
  id INTEGER PRIMARY KEY,
  titel TEXT NOT NULL
    CHECK(length(trim(titel)) > 0),
  jahr INTEGER NOT NULL
    CHECK(jahr BETWEEN 1450 AND 2100),
  preis_cent INTEGER NOT NULL
    CHECK(preis_cent >= 0)
) STRICT;
```

Die Jahresgrenzen sind eine bewusste Vereinfachung für unsere Beispielsammlung, keine Aussage über die gesamte Buchgeschichte. Ein Archiv mit älteren Drucken oder angekündigten Werken muss andere Regeln erhalten. Das zeigt, warum du Regeln fachlich begründen solltest: Eine syntaktisch korrekte Grenze kann für eine andere Sammlung falsch sein.

NOT NULL verhindert einen unbekannten Wert. Die CHECK-Bedingung begrenzt bekannte Werte. Das sind unterschiedliche Aufgaben. Eine CHECK-Bedingung allein weist in SQLite nicht jeden NULL-Ausdruck zurück, weil ein unbekanntes Prüfungsergebnis nicht wie ein eindeutig falsches Ergebnis behandelt wird. Bei einer Pflichtangabe ergänzen wir deshalb ausdrücklich NOT NULL. [D06]

3.4 Unbekannt und leer unterscheiden

NULL beschreibt einen fehlenden oder unbekannten Wert. Eine leere Zeichenfolge ist dagegen ein vorhandener Text mit der Länge null. Null Eurocent sind ein bekannter Geldbetrag. Diese drei Fälle darfst du nicht austauschbar verwenden. Ein unbekannter Kaufpreis ist nicht automatisch ein kostenlos erhaltenes Buch.

Im späteren Ausleihmodell bleibt der tatsächliche Rückgabetag zunächst NULL. Das bedeutet, dass noch keine Rückgabe erfasst wurde. Für einen absichtlich leeren Namen wäre eine leere Zeichenfolge kein guter Ersatz. Wenn das Modell einen Namen verlangt, prüfst du ihn als Pflichttext. Wenn du eine Angabe nicht benötigst, lässt du die Spalte am besten ganz weg.

Beim Filtern auf unbekannte Werte verwendest du IS NULL beziehungsweise IS NOT NULL. Ein Vergleich mit = NULL ergibt nicht die erwartete Trefferliste. SQL arbeitet bei unbekannten Werten mit einer zusätzlichen Wahrheitsmöglichkeit. Gewöhne dir deshalb an, fehlende Werte bereits beim Entwurf zu benennen und später ausdrücklich zu behandeln.

3.5 Regeln schrittweise überprüfen

Füge zunächst genau ein Buch ein. Nenne die Spalten beim INSERT ausdrücklich. Dadurch bleibt verständlich, welcher Wert in welches Feld gehört; zusätzliche Spalten verändern die Bedeutung deiner Werte nicht unbemerkt.

```
INSERT INTO buch
  (id, titel, jahr, preis_cent)
VALUES
  (1, 'Der kleine Garten', 2022, 1290);

SELECT id, titel, jahr, preis_cent
FROM buch;
```

Du solltest eine Zeile mit der Kennung 1 sehen. Versuche danach als eigene Fehlerprobe einen negativen Preis in einer neuen Zeile. Der Befehl muss scheitern. Wenn dein Werkzeug eine Transaktion offen hält, prüfe vor dem Weiterarbeiten, welcher Zustand nach dem Fehler vorliegt. Nicht jede Oberfläche behandelt mehrere Anweisungen und Fehler identisch.

Ein abgewiesener Wert ist ein Erfolg deiner Datenregel. Halte in deiner Notiz fest, welche Bedingung verletzt wurde. Die Regel schützt später auch dann, wenn Daten über eine andere Anwendung oder einen Import kommen. Eine reine Prüfung im Eingabeformular könnte dort leicht umgangen werden.