

Überblick

Unter der **Normalisierung** einer relationalen Datenbank versteht man die Umwandlung einer beliebigen Datenbank in eine bestimmte Form. Dieser Prozess verfolgt eine Reihe von Zielen, unter anderem, dass die Datenbank keine Redundanzen mehr enthält („Redundanz“ bedeutet, dass gleiche Daten mehrfach in Tabellen enthalten sind). Dies trägt zur Vermeidung von Fehlern bei und sorgt außerdem dafür, dass alle Betroffenen sich auf eine einheitliche Struktur der Datenbank verlassen können.

Der Prozess der Normalisierung wurde unter anderen entwickelt von Edgar F. Codd, dem Entwickler des relationalen Datenbankmodells.

Beispieldatenbank

Als Ausgangspunkt betrachten wir eine nicht normalisierte Datenbank. Sie enthält alle Daten in einer Tabelle. Die Daten stellen die Bestellungen eines Schreibwarengroßhandels dar. Im Beispiel hat der Kunde Schmidt am 01.03.2021 200 Blöcke A4 zum Einzelpreis von 0,45 €, 100 Blöcke A5 zum Preis von 0,30 € und 30 Lineale zum Preis von 1,50 € bestellt. Die Tabelle enthält weitere Daten wie Kunden- und Bestellnummern, die Adressen der Kunden sowie die Gesamtbeträge der einzelnen Artikel einer Bestellung.

BestNr	Datum	KdNr	KdName	KdAdresse	Artikel	ArtPreis	Betrag
1001	01.03.2021	K01	Schmidt	Kaiserstr. 5, 53113 Bonn	A10, Block A4, 200 A15, Block A5, 100 A50, Kugelschreiber, 30	0,45 0,30 1,50	90,00 30,00 45,00
1002	01.03.2021	K02	Müller	Händelstr. 67, 53115 Bonn	A60, Radiergummi, 100 A50, Kugelschreiber, 20 A80, Büroklammer, 1000	0,05 1,60 0,01	5,00 32,00 10,00
1003	03.03.2021	K01	Schmidt	Kaiserstr. 5, 53113 Bonn	A10, Block A4, 100 A50, Kugelschreiber, 50	0,50 1,40	50,00 70,00

Eine Besonderheit ergibt sich daraus, dass der Einzelpreis für Artikel nicht immer gleich ist – am 03.03. hat der Kunde Schmidt weitere 100 Blöcke A4 bestellt, die jetzt 0,50 € pro Stück kosten – d.h. bei großen Bestellungen sinkt der Preis der Artikel.

Prozess der Normalisierung: 1NF → 2NF → 3NF

Der Prozess der Normalisierung verläuft in drei Schritten¹: zuerst bildet man die erste Normalform (1NF). Aus der ersten Normalform bildet man die zweite (2NF), und daraus die dritte (3NF). Man kann dabei keinen Schritt auslassen, d.h. um die 2NF zu bilden braucht man die 1NF als Grundlage.

¹ Im Informatikunterricht behandeln wir die ersten drei Normalformen. Es gibt noch weitere Normalformen (BCNF, 4NF und 5NF), die im Unterricht jedoch nicht behandelt werden. Bei Interesse sei z.B. auf den Wikipedia-Artikel zur Normalisierung verwiesen.

Erste Normalform (1NF)

Die **Ziele** der 1NF sind

- **Atomisierung** von Daten:
Alle Tabellenzellen enthalten jeweils nur einen Wert (atomar = nicht mehr teilbar)
- Vermeiden **berechneter** Werte:
Es soll keine Werte geben, die sich aus anderen Werten in der Tabelle errechnen lassen.
- Eindeutiges Kennzeichnen von Datensätzen durch **Primärschlüssel**

Zunächst wird die Ausgangstabelle so umstrukturiert, dass alle Daten atomar sind und es keine berechneten Werte mehr gibt. Anschließend werden Spalten für den Primärschlüssel ausgewählt.

In unserem Beispiel muss man die Adresse der Kunden in mehrere **Spalten** aufteilen (Straße, Hausnummer, Postleitzahl und Ort), und ebenso die Daten zu den Artikeln (Artikelnummer, Artikelname, Bestellmenge). Die Zellen zu den Artikeln und zu den Preisen enthalten außerdem jeweils mehrere Artikel. Diese teilt man in mehrere **Zeilen** auf.

Die Spalte „Betrag“ wird weggelassen, da dieser sich aus Artikelpreis und Bestellmenge berechnet.

BestNr	Datum	KdNr	KdName	KdAdresse	Artikel	ArtPreis	Betrag
1001	01.03.2021	K01	Schmidt	Kaiserstr. 5, 53113 Bonn	A10, Block A4, 200 A15, Block A5, 100 A50, Kugelschreiber, 30	0,45 0,30 1,50	90,00 30,00 45,00
1002	01.03.2021	K02	Müller	Händelstr. 67, 53115 Bonn	A60, Radiergummi, 100 A50, Kugelschreiber, 20 A80, Büroklammer, 1000	0,05 1,60 0,01	5,00 32,00 10,00
1003	03.03.2021	K01	Schmidt	Kaiserstr. 5, 53113 Bonn	A10, Block A4, 100 A50, Kugelschreiber, 50	0,50 1,40	50,00 70,00

Nicht-atomare Daten:
Aufteilen in Spalten

Gleichartige Daten:
Aufteilen in Zeilen

Berechnete Werte:
weglassen

Die 1NF zu unserem Beispiel ist dann die folgende Tabelle:

BestNr	Datum	KdNr	KdName	Straße	Nr	PLZ	Ort	ArtNr	ArtName	Menge	Preis
1001	01.03.2021	K01	Schmidt	Kaiserstr.	5	53113	Bonn	A10	Block A4	200	0,45
1001	01.03.2021	K01	Schmidt	Kaiserstr.	5	53113	Bonn	A15	Block A5	100	0,30
1001	01.03.2021	K01	Schmidt	Kaiserstr.	5	53113	Bonn	A50	Kugelschr.	30	1,50
1002	01.03.2021	K02	Müller	Händelstr.	67	53115	Bonn	A60	Radiergummi	100	0,05
1002	01.03.2021	K02	Müller	Händelstr.	67	53115	Bonn	A50	Kugelschr.	20	1,60
1002	01.03.2021	K02	Müller	Händelstr.	67	53115	Bonn	A80	Büroklammer	1000	0,01
1003	03.03.2021	K01	Schmidt	Kaiserstr.	5	53113	Bonn	A10	Block A4	100	0,50
1003	03.03.2021	K01	Schmidt	Kaiserstr.	5	53113	Bonn	A50	Kugelschr.	50	1,40

Als **Primärschlüssel** wurden Bestellnummer und Artikelnummer ausgewählt, da diese jede Zeile eindeutig kennzeichnen (in einer Bestellung kann der gleiche Artikel nicht mehrmals vorkommen – stattdessen würde man die Menge erhöhen.)

Zur Begründung:

- Die Daten sollen atomar sein, damit man besser **nach einzelnen Daten filtern** kann. So könnte man z.B. alle Kunden suchen, die eine bestimmte Postleitzahl haben.
- Atomare Daten sind **leichter zu verändern**. Wenn z.B. ein Kunde seine Bestellung ändert und statt 30 Linealen jetzt 35 Lineale bestellt, ist es mit SQL wesentlich einfacher, nur den Wert „Bestellmenge“ zu ändern als eine komplexe Tabellenzelle, die mehrere Artikel mit Namen, Nummern und Bestellmenge enthält.
- Berechnete Werte lässt man weg, da sie potentielle **Fehlerquellen** darstellen. Wenn sich z.B. die Bestellmenge für Lineale ändert, müsste sich auch der Gesamtpreis ändern. Das kann leicht vergessen werden, und dann sind die Daten inkonsistent – später ist dann kaum mehr nachzuvollziehen, ob die Bestellmenge oder der Gesamtpreis den richtigen Wert enthält.
- Primärschlüssel sind nötig, um die **weiteren Normalformen** (2NF und 3NF) bilden zu können, bei denen die große Tabelle in mehrere Entitäts- und Beziehungstabellen aufgeteilt wird.

Zweite Normalform (2NF)

Das Hauptziel der 2NF ist die **Vermeidung redundanter Daten**.

In unserer Beispieltabelle tauchen viele Daten mehrmals auf. Der Kunde Schmidt wird z.B. fünf Mal aufgeführt, jeweils mit Namen und allen Adressdaten. Die Wiederholungen sind redundant, d.h. überflüssig – sie enthalten keine neuen Daten.

Um Redundanzen zu vermeiden wird die Tabelle in mehrere Tabellen aufgeteilt.

Dabei orientiert man sich in der 2NF **ausschließlich an den Primärschlüsselspalten**.

1. Schritt: Bestimme zu jeder Spalte, von welcher Primärschlüsselspalte sie funktional abhängig ist.

Funktionale Abhängigkeit bedeutet hier: die Daten einer Spalte beziehen sich direkt auf die Daten einer oder mehrerer Primärschlüsselspalten. Wie man der Tabelle der 1NF auf Seite 2 entnehmen kann, ist das Datum ist zum Beispiel (nur) abhängig von der Bestellnummer, weil zu einer Bestellnummer immer das gleiche Datum angegeben wird. Die Menge ist hingegen von der Kombination aus Bestellnummer und Artikelnummer abhängig, denn zu einer Bestellnummer gibt es verschiedene Mengen, ebenso zu einer Artikelnummer.

Man kann die Abhängigkeiten tabellarisch auflisten:

Spalte	abhängig von	Begründung
Datum	BestNr	Das Datum gehört zur Bestellung und ist unabhängig vom Artikel. Das gleiche gilt für die Kundendaten.
Kundenspalten	BestNr	
ArtName	ArtNr	Der Artikelname gehört zur Artikelnummer.
Menge	BestNr und ArtNr	Jeder Artikel hat pro Bestellung eine andere Menge, und der Preis eines Artikels kann je nach Bestellung variieren.
Preis	BestNr und ArtNr	

2. Schritt: Bilde zu jeder Kombination von Primärschlüsselspalten eine eigene Tabelle, mit den Spalten, die von dieser Kombination abhängig sind.

Im Beispiel werden also eine Tabelle mit dem Primärschlüssel BestNr, eine Tabelle mit ArtNr, und eine Tabelle mit der Kombination BestNr und ArtNr gebildet:

Bestellungen

BestNr	Datum	KdNr	KdName	Straße	Nr	PLZ	Ort
1001	01.03.2021	K01	Schmidt	Kaiserstr.	5	53113	Bonn
1002	01.03.2021	K02	Müller	Händelstr.	67	53115	Bonn
1003	03.03.2021	K01	Schmidt	Kaiserstr.	5	53113	Bonn

Artikel

ArtNr	ArtName
A10	Block A4
A15	Block A5
A50	Kugelschreiber
A60	Radiergummi
A80	Büroklammer

Bestelldaten

↑BestNr	↑ArtNr	Menge	Preis
1001	A10	200	0,45
1001	A15	100	0,30
1001	A50	30	1,50
1002	A60	100	0,05
1002	A50	20	1,60
1002	A80	1000	0,01
1003	A10	100	0,50
1003	A50	50	1,40

Die 2NF entspricht schon eher einer relationalen Datenbank: Wir haben jetzt zwei Entitätstabellen (Bestellungen und Artikel) und eine Beziehungstabelle (Bestelldaten).

Durch die Trennung in mehrere Tabellen fallen viele redundante Daten weg – der Artikelname „Kugelschreiber“ kommt z.B. in der gesamten Datenbank nur noch einmal vor, weil er in den Bestelldaten nicht jedes Mal aufgeführt werden muss – dort reicht als Fremdschlüssel die Artikelnummer.

Zur Begründung:

- Redundante Daten führen leicht zu Fehlern, den sog. **Änderungsanomalien**.
Wenn sich z.B. die Adresse des Kunden Schmidt ändert, muss das an vielen Stellen geändert werden. Vergisst man einen Datensatz, enthält die Datenbank inkonsistente Daten. Es ist dann schwer nachzuvollziehen, welche Adresse die richtige ist.
- Wenn Daten, die eigentlich unabhängig voneinander sind, in einer einzigen Tabelle enthalten sind, kann es zu **Löschanomalien** kommen. In der 1NF ist der Artikel „Radiergummi“ nur einmal enthalten. Wenn der Kunde sich entschließt, diesen Artikel zu stornieren, wird die Zeile aus der Tabelle entfernt. Dann ist dieser Artikel jedoch gar nicht mehr in der Datenbank enthalten! In der 2NF kann das nicht passieren: Der Kunde kann den Artikel aus seinen Bestelldaten entfernen, er bleibt in der Artikeltabelle aber erhalten.

Dritte Normalform (3NF)

Das Hauptziel der 3NF ist, wie bei der 2NF, die Vermeidung redundanter Daten.

Auch in der 2NF sind noch redundante Daten enthalten – die Daten zum Kunden Schmidt gibt es in der Tabelle der Bestellungen immer noch zwei Mal. Um das zu vermeiden, werden die Tabellen der 2NF noch weiter aufgeteilt.

Dabei orientiert man sich in der 3NF **ausschließlich an Nicht-Schlüsselspalten**.

1. Schritt: Bestimme Spalten, die von Nichtschlüsselspalten abhängig sind

Spalte	abhängig von	Begründung
KdName, Straße, Nr, PLZ, Ort,	KdNr	Die Adressdaten sind nur von der Kundennummer abhängig, nicht von Bestellungen oder Artikeln.

2. Schritt: Bilde für diese Nicht-Schlüsselspalten weitere Tabellen, mit deren abhängigen Spalten.

Bestellungen

<u>BestNr</u>	Datum	↑KdNr
1001	01.03.2021	K01
1002	01.03.2021	K02
1003	03.03.2021	K01

Kunden

<u>KdNr</u>	KdName	Straße	Nr	PLZ	Ort
K01	Schmidt	Kaiserstr.	5	53113	Bonn
K02	Müller	Händelstr.	67	53115	Bonn

Artikel

<u>ArtNr</u>	ArtName
A10	Block A4
A15	Block A5
A50	Kugelschreiber
A60	Radiergummi
A80	Büroklammer

Bestelldaten

↑ <u>BestNr</u>	↑ <u>ArtNr</u>	Menge	Preis
1001	A10	200	0,45
1001	A15	100	0,30
1001	A50	30	1,50
1002	A60	100	0,05
1002	A50	20	1,60
1002	A80	1000	0,01
1003	A10	100	0,50
1003	A50	50	1,40

Die **Gründe** für die 3NF sind die gleichen wie für die 2NF:
Das Vermeiden von Änderungs- und Löschanomalien.

Vor- und Nachteile der Normalisierung

Die **Vorteile** der Normalisierung liegen auf der Hand und wurden in den vorigen Abschnitten erläutert: eine einheitliche Struktur für Datenbanken, Vermeidung von Redundanzen und damit Fehlerquellen usw.

Die Normalisierung von Datenbanken hat jedoch auch einen entscheidenden **Nachteil**, der zunächst nicht so offensichtlich ist. Und zwar führt die Teilung einer Datenbank in viele Tabellen zu einer Verminderung der **Performance von SQL-Abfragen**: Bestimmte SQL-Abfragen benötigen wesentlich mehr Zeit, Rechenleistung und Speicher, wenn eine Datenbank normalisiert ist.

Das liegt daran, dass Verbünde mit SQL aufwendig sind. Für jeden JOIN müssen die Zeilen einer Tabelle mit jeder Zeile einer anderen Tabelle verglichen werden, entsprechend dem Kreuzprodukt, das im Abschnitt zum Verbund erläutert wurde. Ein Verbund über zwei Tabellen mit je 1000 Zeilen benötigt daher schon eine Million Vergleiche.

Wenn eine Datenbank aus vielen Tabellen besteht, enthalten SELECT-Abfragen oft JOINS über mehrere Tabellen, und das Kreuzprodukt muss dabei für jede Abfrage mehrmals gebildet werden.

In der Praxis kann das dazu führen, dass aufwendige SQL-Skripte für große Datenbanken unvertretbar viel Zeit benötigen. Wenn man eine tägliche Auswertung einer Business-Datenbank benötigt, um z.B. die Tageseinnahmen zu berechnen, und die Berechnung länger als 24 Stunden dauert, ist die Datenbank praktisch nicht mehr einsetzbar.

Die Datenbank muss dann „de-normalisiert“ werden. Dabei wird die Zahl der Tabellen verringert, wobei man in Kauf nimmt, dass sich bestimmte Daten in mehreren Tabellen wiederholen. Dafür sind in den SQL-Skripten dann weniger JOINS notwendig.