Normalformen

9

In einem schlecht gewählten Datenbankschema können Einfüge, Änderungs- und Löschoperationen zu gewissen Anomalien führen. Es kann beispielsweise sein, dass für eine Änderung mehrere Änderungsoperationen ausgeführt werden müssen oder dass gewisse Daten gar nicht im gegebenen Schema abgespeichert werden können.

Einfach gesagt treten diese Anomalien immer dann auf, wenn ein Schema mehrere Konzepte modelliert. Um dies formal zu präzisieren, führen wir den Begriff der funktionalen Abhängigkeit ein. Dies hilft uns, Relationenschemata so zu zerlegen, dass ein Schema nur noch ein Konzept modelliert und damit Anomalien vermieden werden. Wir sagen dann, das Schema ist in Normalform. In diesem Kapitel studieren wir insbesondere die dritte Normalform (3NF) und die Boyce–Codd Normalform (BCNF). Dabei interessiert uns, ob eine Zerlegung in eine Normalform verlustfrei und abhängigkeitserhaltend ist. Auch geben wir den Zusammenhang zwischen 3NF und transitiven Abhängigkeiten an.

9.1 Anomalien

Wir beginnen mit einem Beispiel zu einer Bibliotheksdatenbank.

Beispiel 9.1. Unsere Datenbank zur Bibliotheksverwaltung besitzt zwei Schemata: $\mathscr{B}$ modelliert den Buchbestand der Bibliothek, $\mathscr{A}$ modelliert die Ausleihen:

$$\mathscr{B} := (\underline{\texttt{BId}}, \texttt{ISBN}, \texttt{Titel}, \texttt{Autor})$$

$$\mathscr{A} := (\underline{\texttt{BId}}, \texttt{Name}, \texttt{Adresse}, \texttt{Datum})$$

Das Attribut `BId` ist eine eindeutige Id für die Bücher, welche zum Bestand der Bibliothek gehören. Für jedes dieser Bücher wird die ISBN, der Titel und der Autor abgespeichert.

T. Studer, *Relationale Datenbanken*,
https://doi.org/10.1007/978-3-662-58976-2_9

Bei einer Ausleihe wird die Id des ausgeliehenen Buches, der Name und die Adresse des Benutzers, welcher das Buch ausleiht, sowie das Ausleihdatum abgespeichert. Wir nehmen hier an, dass ein Benutzer eindeutig durch seinen Namen identifiziert ist. Ausserdem sei `BId` im Schema $\mathscr{A}$ ein Fremdschlüssel auf das Schema $\mathscr{B}$. Damit können nur Bücher ausgeliehen werden, welche zum Bestand der Bibliothek gehören.

Dieses Schema hat den Vorteil, dass für jede Ausleihe unmittelbar auf die entsprechenden Benutzerdaten zugegriffen werden kann. Dem stehen jedoch auch einige Nachteile gegenüber.

Änderungsanomalie (Update-Anomalie) Nehmen wir an, eine Benutzeradresse soll geändert werden. Falls dieser Benutzer mehrere Bücher ausgeliehen hat, so gibt es mehrere Einträge mit diesem Benutzer in der $\mathscr{A}$-Relation. In jedem dieser Einträge muss nun die Adresse aktualisiert werden. Das heisst, obwohl nur die Adresse *eines* Benutzers ändert, müssen *mehrere* Tupel aktualisiert werden. Der Grund dafür ist natürlich die Redundanz, das mehrfache Vorhandensein, der Daten eines Bibliotheksbenutzers. Dem Vorteil der einfachen Datenbankabfrage steht also der Nachteil komplexer Änderungsoperationen gegenüber.

Einfügeanomalie (Insertion-Anomalie) Ein neuer Benutzer kann nur erfasst werden, falls er auch zugleich ein Buch ausleiht. Wenn sich jemand neu anmeldet ohne ein Buch auszuleihen, dann können seine Daten (Name und Adresse) nicht in der Datenbank eingetragen werden.

Löschanomalie (Deletion-Anomalie) Wenn ein Benutzer alle ausgeliehenen Bücher zurückbringt, dann werden alle Daten über diesen Benutzer in der $\mathscr{A}$-Relation gelöscht. Es sind damit keine Informationen mehr über ihn gespeichert. Somit müssen bei einer neuen Ausleihe alle Benutzerdaten wieder neu erfasst werden.

Der Grund für diese Anomalien ist, dass durch *ein* Schema *mehrere* Konzepte modelliert werden. Tatsächlich werden im Schema $\mathscr{A}$ sowohl Informationen über die Ausleihen als auch über die Benutzer abgespeichert. Auch das Schema $\mathscr{B}$ enthält Informationen über zwei Konzepte: nämlich den aktuellen Buchbestand und die Buchausgaben.

Um Anomalien zu vermeiden, brauchen wir also eine formale Beschreibung dieser Sachverhalte. Dazu ist es nützlich zu studieren, welche Attribute von welchen anderen Attributen abhängig sind. Beispielsweise sehen wir, dass im Schema $\mathscr{B}$ der Wert des Attributes `Adresse` vom Wert des Attributes `Name` abhängig ist.

Im nächsten Abschnitt werden wir solche Abhängigkeiten präzise einführen und studieren. Dies dient später dazu, Normalformen für Schemata zu definieren. Man kann auf diese Weise zeigen, dass gewisse Anomalien nicht auftreten können, falls ein Schema in Normalform ist.

9.2 Funktionale Abhängigkeiten

Definition 9.2. Es seien $A_1, \ldots, A_n$ Attribute. Eine *funktionale Abhängigkeit* (functional dependency) auf einer Attributmenge $\{A_1, \ldots, A_n\}$ ist gegeben durch

$$X \to Y,$$

wobei $X, Y \subseteq \{A_1, \ldots, A_n\}$. Wir verwenden die Sprechweise *eine funktionale Abhängigkeit auf einem Schema* als Abkürzung für eine funktionale Abhängigkeit auf der Menge der Attribute des Schemas.

Eine funktionale Abhängigkeit $X \to Y$ heisst *trivial*, falls $Y \subseteq X$ gilt.

Um die Bedeutung von funktionalen Abhängigkeit formal einzuführen, benötigen wir folgende Notation. Sei R eine Relation über den Attributen $A_1, \ldots, A_n$. Ferner sei $X \subseteq \{A_1, \ldots, A_n\}$. Für $s, t \in R$ schreiben wir

$$s[X] = t[X],$$

falls

$$s[A_i] = t[A_i] \text{ für alle } A_i \in X$$

gilt.[1]

Definition 9.3 (Erfüllung funktionaler Abhängigkeiten). Gegeben sei eine Relation R eines Schemas $\mathscr{S}$. Ferner sei $X \to Y$ eine funktionale Abhängigkeit auf $\mathscr{S}$. Die Relation R *erfüllt* $X \to Y$, falls für alle Tupel $s, t \in R$ gilt

$$s[X] = t[X] \implies s[Y] = t[Y]. \tag{9.1}$$

Eine funktionale Abhängigkeit $X \to Y$ drückt aus, dass wenn zwei Tupel s und t auf allen Attributen aus X übereinstimmen, so müssen s und t auch auf allen Attributen aus Y übereinstimmen. Oder anders ausgedrückt, wenn die Werte auf den X-Attributen gegeben sind, dann sind die Werte auf den Y-Attributen eindeutig bestimmt.

Anmerkung 9.4. Wir wollen nun kurz auf das Problem von `Null` Werten im Zusammenhang mit funktionalen Abhängigkeiten eingehen. Im Beispiel 2.4 haben wir gesehen,

[1] Der Unterschied zur Notation in (2.2) besteht darin, dass X hier eine Menge ist und nicht eine Sequenz. Damit können wir $s[X]$ hier nicht als eigenständiges Tupel verwenden.

dass (9.1) nicht erfüllt werden kann, falls die vorkommenden Tupel `Null` Werte in den Attributen aus Y enthalten.

Wir könnten dieses Problem lösen, indem wir anstelle von (9.1) die Bedingung

$$s[X] = t[X] \quad \Longrightarrow \quad s[Y] \simeq t[Y] \tag{9.2}$$

verwenden. Dann haben wir jedoch das Problem, dass `Null` Werte in den Attributen aus X problematisch sind, siehe Beispiel 2.7. In Anmerkung 10.4 werden wir noch im Detail zeigen, was mit (9.2) schief geht.

Um diese Probleme mit `Null` Werten zu vermeiden, treffen wir nun folgende Annahme.

Annahme 9.5 *In diesem und dem nächsten Kapitel enthalten alle vorkommenden Datenbanken keine* `Null` *Werte.*

Anmerkung 9.6. Gegeben seien ein Schema $\mathscr{S}$ und eine *triviale* funktionale Abhängigkeit $X \to Y$ auf $\mathscr{S}$. Offensichtlich erfüllt jede Relation auf $\mathscr{S}$ die funktionale Abhängigkeit $X \to Y$.

Beispiel 9.7. Betrachte folgende Relation `Ausleihen` über dem Schema $\mathscr{A}$ aus Beispiel 9.1:

Ausleihen

BId	Name	Adresse	Datum
11	Eva	Thun	20140506
5	Eva	Thun	20140804
4	Tom	Bern	20140301

Die Relation `Ausleihen` erfüllt die funktionalen Abhängigkeiten

$$\{\texttt{BId}\} \to \{\texttt{Name}, \texttt{Adresse}, \texttt{Datum}\} \tag{9.3}$$

und

$$\{\texttt{Name}\} \to \{\texttt{Adresse}\}. \tag{9.4}$$

Die Abhängigkeit (9.3) drückt aus, dass der Wert des Attributs `BId` eindeutig ein Tupel der Relation identifiziert. Die Abhängigkeit (9.4) sagt, dass Benutzer eindeutig durch ihren Namen identifiziert werden.

Folgende funktionale Abhängigkeit ist in `Ausleihen` verletzt, d. h. sie ist *nicht* erfüllt:

$$\{\texttt{Name}\} \to \{\texttt{Datum}\}.$$

Dies zeigt sich daran, dass es zwei Tupel mit demselben Wert im Attribut `Name` aber verschiedenen Werten im Attribut `Datum` gibt.

Anmerkung 9.8. Wir können eine funktionale Abhängigkeit $X \rightarrow Y$ über einem Schema $\mathscr{S}$ als Constraint betrachten. Die entsprechende Integritätsregel verlangt dann, dass jede Instanz R von $\mathscr{S}$ die Abhängigkeit $X \rightarrow Y$ erfüllen muss.

In diesem Sinne können wir unique Constraints als spezielle funktionale Abhängigkeiten darstellen (unter der Annahme, dass keine `Null` Werte auftreten). Gegeben sei DB-Schema $\mathscr{S} = (A_1, \ldots, A_n)$. Ein Unique constraint

$$U = (A_{i_1}, \ldots, A_{i_m})$$

kann durch folgende funktionale Abhängigkeit ausgedrückt werden:

$$\{A_{i_1}, \ldots, A_{i_m}\} \rightarrow \{A_1, \ldots, A_n\}.$$

Definition 9.9. Wir führen nun eine Reihe von Abkürzungen ein um die Notation im Zusammenhang mit funktionalen Abhängigkeiten zu vereinfachen. Es sei

$$\mathscr{S} = (A_1, \ldots, A_n)$$

ein Schema und $X, Y, Z \subseteq \{A_1, \ldots, A_n\}$.

1. Wir verwenden YZ für $Y \cup Z$. Somit steht beispielsweise

$$X \rightarrow YZ \quad \text{für} \quad X \rightarrow Y \cup Z.$$

2. Wir schreiben $\mathscr{S}$ für $\{A_1, \ldots, A_n\}$. Damit betrachten wir $\mathscr{S}$ als ungeordnete Menge und

$$X \rightarrow \mathscr{S} \quad \text{steht für} \quad X \rightarrow \{A_1, \ldots, A_n\}.$$

3. Wir verwenden A_i für die einelementige Menge $\{A_i\}$. Somit steht beispielsweise

$$X \rightarrow A_i \quad \text{für} \quad X \rightarrow \{A_i\}.$$

   Damit können wir auch

$$X \rightarrow A_i A_j \quad \text{für} \quad X \rightarrow \{A_i, A_j\}$$

   schreiben.

Definition 9.10 (Logische Folgerung). Gegeben sei ein Schema $\mathscr{S} = (A_1, \ldots, A_n)$. Ferner seien eine Menge F von funktionalen Abhängigkeiten über $\mathscr{S}$ sowie eine weitere funktionale Abhängigkeit $X \to Y$ über $\mathscr{S}$ gegeben.

Wir sagen, $X \to Y$ *folgt logisch aus* F, in Zeichen

$$F \models X \to Y,$$

falls jede Instanz R von $\mathscr{S}$, welche alle Abhängigkeiten in F erfüllt, auch $X \to Y$ erfüllt.

Beispiel 9.11. Gegeben sei die Menge $F = \{W \to X,\ W \to Y,\ XY \to Z\}$ von funktionalen Abhängigkeiten. Dann gilt unter anderem:

1. $F \models W \to X$,
2. $F \models W \to XY$,
3. $F \models W \to Z$.

Diese drei Eigenschaften ergeben sich unmittelbar aufgrund von Definitions 9.3 und 9.10 durch Betrachtung der auftretenden Tupel.

Definition 9.12 (Hülle F^+). Ist F eine Menge von funktionalen Abhängigkeiten, so wird die *Hülle* F^+ von F definiert durch

$$F^+ := \{X \to Y \mid F \models X \to Y\}.$$

Wir betrachten nun funktionale Abhängigkeiten wieder als Integritätsbedingungen eines DB-Schemas. Sei $\mathscr{S} = (A_1, \ldots, A_n)$ ein Schema mit einer Menge von funktionalen Abhängigkeiten F. Falls F eine nicht-triviale funktionale Abhängigkeit enthält, dann gibt es eine echte Teilmenge X von $\{A_1, \ldots, A_n\}$, so dass

$$X \to \{A_1, \ldots, A_n\} \in F^+.$$

Wir sind an möglichst kleinen derartigen Teilmengen interessiert, da jedes Tupel einer Instanz von $\mathscr{S}$ durch die Werte in den Attributen dieser Teilmenge eindeutig identifiziert ist.

Definition 9.13 (Schlüssel). Gegeben sei ein Schema $\mathscr{S} = (A_1, \ldots, A_n)$ mit einer Menge von funktionalen Abhängigkeiten F. Eine Teilmenge

$$X \subseteq \{A_1, A_2, \ldots, A_n\}$$

heisst *Superschlüssel* für $\mathscr{S}$ bezüglich F, falls gilt:

$$X \to \mathscr{S} \in F^+.$$

Ein Superschlüssel für $\mathscr{S}$ heisst *Schlüssel* für $\mathscr{S}$ bezüglich F, falls zusätzlich gilt:

$$\text{es gibt keine } \textit{echte} \text{ Teilmenge } Y \subsetneq X \text{ mit } Y \to \mathscr{S} \in F^+.$$

Beispiel 9.14. Wir betrachten Postleitzahlen und treffen folgende vereinfachenden Annahmen:

1. Jede Stadt ist eindeutig durch ihre Postleitzahl bestimmt. Das heißt es gibt keine zwei Städte mit derselben Postleitzahl.
2. Jede Postleitzahl ist eindeutig durch Stadt und Strasse bestimmt. Das heißt die Postleitzahl ändert sich innerhalb einer Strasse nicht.

Entsprechend wählen wir Attribute `Stadt`, `Str` und `PLZ` sowie das Schema

$$\mathscr{S} = (\texttt{Stadt}, \texttt{Str}, \texttt{PLZ})$$

mit der dazugehörigen Menge

$$\big\{ \{\texttt{Stadt}, \texttt{Str}\} \to \{\texttt{PLZ}\}, \quad \{\texttt{PLZ}\} \to \{\texttt{Stadt}\} \big\}$$

von funktionalen Abhängigkeiten. Mit der obigen Definition folgt sofort, dass die Attributmengen {`Stadt`, `Str`} und {`Str`, `PLZ`} Schlüssel von $\mathscr{S}$ sind.

9.3 Zerlegung von Relationenschemata

Wir betrachten eine Strategie, um schlechte Schemata durch geeignete Zerlegungen zu verbessern. Ein Ansatz besteht darin jedes Schema, das auf vielen Attributen basiert, durch mehrere Schemata mit jeweils weniger Attributen zu ersetzen. Zur Erinnerung: Alle vorkommenden Datenbanken enthalten keine `Null` Werte (Annahme 9.5).

Definition 9.15. Gegeben seien die Attribute $A_1, A_2, \ldots, A_n$ sowie das Schema

$$\mathscr{S} := (A_1, \ldots, A_n).$$

Eine *Zerlegung* von $\mathscr{S}$ ist eine Menge von Relationenschemata

$$\{ (A_{11}, A_{12}, \ldots, A_{1m_1}), \ldots, (A_{k1}, A_{k2}, \ldots, A_{km_k}) \},$$

so dass gilt

$$\{A_{11}, A_{12}, \ldots, A_{1m_1}\} \cup \cdots \cup \{A_{k1}, A_{k2}, \ldots, A_{km_k}\} \;=\; \{A_1, A_2, \ldots, A_n\}.$$

Bei einer Zerlegung dürfen die Schemata, in die das Ausgangsschema zerlegt wird, gemeinsame Attribute besitzen.

Die Zerlegung eines Schemas soll dazu dienen, Anomalien zu vermeiden. Wir haben im Abschn. 9.1 gesehen, dass Anomalien auftreten, wenn ein Schema mehrere Konzepte modelliert. Wir müssen also ein gegebenes Schema so zerlegen, dass jedes Teilschema möglichst nur noch ein Konzept beschreibt.

Beispiel 9.16. Wir betrachten das Schema

$$\mathscr{A} := (\texttt{BId}, \texttt{Name}, \texttt{Adresse}, \texttt{Datum})$$

aus Beispiel 9.1, welches die Ausleihen und die Benutzer modelliert. Wir können dieses Schema in Teile $\mathscr{A}_1$ und $\mathscr{A}_2$ zerlegen, die wie folgt gegeben sind:

$$\mathscr{A}_1 := (\texttt{BId}, \texttt{Name}, \texttt{Datum})$$

$$\mathscr{A}_2 := (\texttt{Name}, \texttt{Adresse})$$

Das Schema $\mathscr{A}_1$ modelliert somit nur noch die Ausleihen und das Schema $\mathscr{A}_2$ entsprechend die Benutzer. In dieser Zerlegung sind die Benutzerdaten nicht mehr redundant gespeichert und es können auch Benutzer verwaltet werden, die kein Buch ausgeliehen haben. Somit sind die Anomalien aus Abschn. 9.1 nicht mehr möglich.

Zwei wichtige Eigenschaften von Zerlegungen sind:

1. die ursprüngliche Information soll aus der Zerlegung wieder rekonstruierbar sein;
2. auf der Zerlegung sollen dieselben funktionalen Abhängigkeiten gelten wie auf dem Ursprungsschema.

Vorerst führen wir folgende Schreibweisen ein. Sei $\mathscr{S}$ das Relationenschema $(A_1, \ldots, A_n)$. Zur Vereinfachung der Notation schreiben wir im Folgenden häufig

$$\pi_{\mathscr{S}}(R) \quad \text{anstelle von} \quad \pi_{A_1,\ldots,A_n}(R).$$

Weiter werden wir die Gleichheit von Relationen bezüglich eines Schemas $\mathscr{S}$ verwenden. Wir setzen:

$$R \;=_{\mathscr{S}}\; T \quad :\Longleftrightarrow \quad \pi_{\mathscr{S}}(R) = \pi_{\mathscr{S}}(T).$$

Somit heisst $R =_{\mathscr{S}} T$, dass die Relationen R und T bezüglich der Attribute aus $\mathscr{S}$ dieselbe Information enthalten.

Beispiel 9.17. Betrachten wir die Attribute `Name`, `Marke` und `Farbe`, sowie das Schema

$$\mathscr{S} := (\texttt{Name}, \texttt{Marke}, \texttt{Farbe})$$

mit der Zerlegung $\{\mathscr{S}_1, \mathscr{S}_2\}$ gegeben durch

$$\mathscr{S}_1 := (\texttt{Name}, \texttt{Marke}) \text{ und } \mathscr{S}_2 := (\texttt{Marke}, \texttt{Farbe}).$$

Ausserdem sei die Instanz `Autos` von $\mathscr{S}$ gegeben durch:

Autos

Name	**Marke**	**Farbe**
Eva	Audi	schwarz
Tom	Audi	rot

Es gilt dann:

$\pi_{\mathscr{S}_1}$ **(Autos)**

Name	**Marke**
Eva	Audi
Tom	Audi

$\pi_{\mathscr{S}_2}$ **(Autos)**

Marke	**Farbe**
Audi	schwarz
Audi	rot

Damit erhalten wir:

$\pi_{\mathscr{S}_1}$ **(Autos)** $\bowtie$ $\pi_{\mathscr{S}_2}$ **(Autos)**

Marke	**Name**	**Farbe**
Audi	Eva	schwarz
Audi	Eva	rot
Audi	Tom	schwarz
Audi	Tom	rot

Es gilt also

$$\pi_{\mathscr{S}_1}(\texttt{Autos}) \bowtie \pi_{\mathscr{S}_2}(\texttt{Autos}) \neq_{\mathscr{S}} \texttt{Autos}.$$

Die ursprüngliche Relation kann also nicht durch einen Verbund aus den Projektionen zurückgewonnen werden. Im konkreten Beispiel ging durch die Zerlegung folgende Information verloren:

1. Eva fährt ein schwarzes Auto,
2. Tom fährt ein rotes Auto.

Das heisst, diese Information kann aus den Daten der Zerlegung nicht mehr herausgelesen werden.

Eine gute Zerlegung sollte keinen Informationsverlust zur Folge haben. Dies führt uns zur nächsten Definition.

Definition 9.18 (Verlustfreie Zerlegungen). Wir gehen aus von einem Schema $\mathscr{S}$, einer Zerlegung $\{\mathscr{S}_1, \ldots, \mathscr{S}_k\}$ von $\mathscr{S}$ sowie einer Menge F von funktionalen Abhängigkeiten über den Attributen von $\mathscr{S}$. Dann besitzt die Zerlegung $\{\mathscr{S}_1, \ldots, \mathscr{S}_k\}$ einen *verlustfreien Verbund bezüglich* F, falls für alle Instanzen R von $\mathscr{S}$, die F erfüllen, gilt, dass

$$R =_{\mathscr{S}} \pi_{\mathscr{S}_1}(R) \bowtie \cdots \bowtie \pi_{\mathscr{S}_k}(R).$$

In diesem Fall sprechen wir von einer *verlustfreien Zerlegung des Schemas* $\mathscr{S}$ *bezüglich* F.

Damit ist im Fall einer verlustfreien Zerlegung (bezüglich F) also die Wiedergewinnung der ursprünglichen Information aus der Zerlegung möglich. Durch geeignete Zerlegungen kann man in vielen Fällen Redundanzen eliminieren oder zumindest ihre Zahl verringern. Andererseits sind jedoch bei Zerlegungen im Allgemeinen mehr Joins zur Beantwortung einer Abfrage erforderlich.

Wir geben nun ein Kriterium dafür an, dass eine Zerlegung in zwei Schemata verlustfrei ist (ohne Beweis). Dazu führen wir folgende Schreibweise ein. Sei $\mathscr{S}$ das Schema $(A_1, \ldots, A_n)$. Dann verwenden wir die Bezeichnung $\mathscr{S}$ auch für die Menge $\{A_1, A_2, \ldots, A_n\}$.

Lemma 9.19. *Gegeben sei ein Schema* $\mathscr{S}$ *mit einer Menge* F *von funktionalen Abhängigkeiten. Eine Zerlegung* $\{\mathscr{S}_1, \mathscr{S}_2\}$ *von* $\mathscr{S}$ *ist genau dann verlustfrei bezüglich* F*, wenn*

1. $\mathscr{S}_1 \cap \mathscr{S}_2 \to \mathscr{S}_1 \in F^+$ *oder*
2. $\mathscr{S}_1 \cap \mathscr{S}_2 \to \mathscr{S}_2 \in F^+$.

Beispiel 9.20. Wir betrachten die Zerlegung unseres Bibliotheksschemas aus Beispiel 9.16. Das Schema

$$\mathscr{A} := (\texttt{BId}, \texttt{Name}, \texttt{Adresse}, \texttt{Datum})$$

wird zerlegt in $\{\mathscr{A}_1, \mathscr{A}_2\}$ mit

$$\mathscr{A}_1 := (\texttt{BId}, \texttt{Name}, \texttt{Datum})$$
$$\mathscr{A}_2 := (\texttt{Name}, \texttt{Adresse}).$$

Zu $\mathscr{A}$ gehört die Menge von funktionalen Abhängigkeiten

$$F := \big\{\{\texttt{BId}\} \to \{\texttt{Name}, \texttt{Adresse}, \texttt{Datum}\}, \{\texttt{Name}\} \to \{\texttt{Adresse}\}\big\}.$$

Wir finden $\mathscr{A}_1 \cap \mathscr{A}_2 = \{\texttt{Name}\}$. Also gilt $\mathscr{A}_1 \cap \mathscr{A}_2 \to \mathscr{A}_2 \in F^+$. Mit Lemma 9.19 folgt also, dass die Zerlegung verlustfrei ist.

Eine weitere wünschenswerte Eigenschaft von Zerlegungen ist, dass die funktionalen Abhängigkeiten erhalten bleiben.

Definition 9.21 (Abhängigkeitserhaltende Zerlegung). Wir betrachten wieder ein Schema $\mathscr{S}$, eine Zerlegung $\{\mathscr{S}_1, \ldots, \mathscr{S}_k\}$ von $\mathscr{S}$ sowie eine Menge F von funktionalen Abhängigkeiten über den Attributen von $\mathscr{S}$.

1. Die *Projektion von F auf eine Attributmenge Z* wird definiert durch
$$\Pi_Z(F) := \{ X \to Y \in F^+ \mid XY \subseteq Z \}.$$
2. Ist $\mathscr{T}$ das Relationenschema $(A_1, \ldots, A_n)$, so setzen wir
$$\Pi_{\mathscr{T}}(F) := \Pi_{\{A_1,\ldots,A_n\}}(F).$$
3. Die Zerlegung $\{\mathscr{S}_1, \ldots, \mathscr{S}_k\}$ heisst *abhängigkeitserhaltende Zerlegung von $\mathscr{S}$ bezüglich F*, falls gilt
$$\Big(\bigcup_{i=1}^{k} \Pi_{\mathscr{S}_i}(F)\Big)^+ = F^+.$$

In der Gleichung des dritten Teils dieser Definition ist die Bedingung

$$\Big(\bigcup_{i=1}^{k} \Pi_{\mathscr{S}_i}(F)\Big)^+ \subseteq F^+$$

trivialerweise immer erfüllt.

Wird ein Relationenschema $\mathscr{S}$ durch eine Zerlegung dargestellt, die nicht abhängigkeitserhaltend bezüglich F ist, so können Updates in Instanzen der Zerlegungen das gegebene F über $\mathscr{S}$ verletzen.

Beispiel 9.22. Wir setzen nun unsere Überlegungen aus Beispiel 9.14 fort. Wir arbeiten also wieder mit den Attributen Stadt, Str und PLZ, sowie den Relationenschemata

$$\mathscr{S} = (\texttt{Stadt}, \texttt{Str}, \texttt{PLZ}),$$
$$\mathscr{S}_1 = (\texttt{Str}, \texttt{PLZ}), \qquad \mathscr{S}_2 = (\texttt{Stadt}, \texttt{PLZ}).$$

Wie vorher sei ausserdem F unsere Menge

$$\{\{\texttt{Stadt}, \texttt{Str}\} \to \texttt{PLZ}, \quad \texttt{PLZ} \to \texttt{Stadt}\}$$

von funktionalen Abhängigkeiten. Mit Lemma 9.19 finden wir, dass $\{\mathscr{S}_1, \mathscr{S}_2\}$ eine verlustfreie Zerlegung von $\mathscr{S}$ bezüglich F ist. Ferner gilt:

$$\Pi_{\mathscr{S}_1}(F) = \left\{ X \to Y \mid XY \subseteq \{\texttt{Str}, \texttt{PLZ}\} \text{ und } \emptyset \models X \to Y \right\},$$
$$\Pi_{\mathscr{S}_2}(F) = \left\{ X \to Y \mid XY \subseteq \{\texttt{Stadt}, \texttt{PLZ}\} \text{ und } \{\texttt{PLZ} \to \texttt{Stadt}\} \models X \to Y \right\}.$$

Jedoch haben wir

$$\Pi_{\mathscr{S}_1}(F) \cup \Pi_{\mathscr{S}_2}(F) \not\models \{\texttt{Stadt}, \texttt{Str}\} \to \texttt{PLZ}.$$

In der Tat, folgende Instanzen S1 von $\mathscr{S}_1$ und S2 von $\mathscr{S}_2$ zeigen, dass F durch die Zerlegung $\{\mathscr{S}_1, \mathscr{S}_2\}$ nicht erhalten wird:

S1

Str	PLZ
Baumstr	2500
Baumstr	2502

S2

Stadt	PLZ
Biel	2500
Biel	2502

Damit folgt nämlich

S1 ⋈ S2

PLZ	Str	Stadt
2500	Baumstr	Biel
2502	Baumstr	Biel

Die Relationen S1 und S2 erfüllen jeweils die projizierten funktionalen Abhängigkeiten $\Pi_{\mathscr{S}_1}(F)$ und $\Pi_{\mathscr{S}_2}(F)$. Der Verbund S1 ⋈ S2 verletzt jedoch die funktionale Abhängigkeit $\{\texttt{Stadt}, \texttt{Str}\} \to \texttt{PLZ}$.

9.4 1NF bis BCNF

Eine Menge von funktionalen Abhängigkeiten kann dazu verwendet werden, um beim Design eines DB-Systems einige der angesprochenen Anomalien zu vermeiden. Bauen wir ein solches System auf, so kann es notwendig werden, eine Relation in mehrere kleinere Relationen zu zerlegen. Unter Ausnutzung funktionaler Abhängigkeiten kann man verschiedene *Normalformen* definieren, die zu einem *guten* DB-Design führen.

Um später die Normalformen präzise zu beschreiben, benötigen wir die folgenden Konzepte. Zur Erinnerung: die Begriffe *Schlüssel* und *Superschlüssel* wurden in Definition 9.13 eingeführt.

Definition 9.23. Gegeben sei ein Relationenschema $\mathscr{S}$ sowie eine Menge F von funktionalen Abhängigkeiten bezüglich $\mathscr{S}$.

1. Ein Attribut A von $\mathscr{S}$ heisst *prim*, falls A Teil eines Schlüssels von $\mathscr{S}$ ist; anderenfalls heisst A *nicht-prim*.
2. Es gelte $X \to Y \in F^+$ und $Y \to X \notin F^+$; ausserdem sei A ein Attribut von $\mathscr{S}$, das weder in X noch in Y vorkommt und für das $Y \to A \in F^+$ gilt. Dann sagen wir, dass A von X *transitiv bezüglich F abhängig* ist.
3. Eine funktionale Abhängigkeit $X \to Y$ mit Attributen von $\mathscr{S}$ heisst *partielle Abhängigkeit* bezüglich F, falls es eine echte Teilmenge Z von X gibt, so dass $Z \to Y \in F^+$ ist. Dann sagen wir, dass Y von X *partiell bezüglich F abhängig* ist.

Beispiel 9.24 (Transitive Abhängigkeit). Wir betrachten wieder das Schema

$$\mathscr{A} := (\texttt{BId}, \texttt{Name}, \texttt{Adresse}, \texttt{Datum})$$

aus Beispiel 9.1 mit den funktionalen Abhängigkeiten

$$F := \big\{\{\texttt{BId}\} \to \{\texttt{Name}, \texttt{Adresse}, \texttt{Datum}\}, \{\texttt{Name}\} \to \{\texttt{Adresse}\}\big\}.$$

Wir haben

$$\texttt{BId} \to \texttt{Name} \in F^+ \text{ und } \texttt{Name} \to \texttt{BId} \notin F^+.$$

Ausserdem gilt

$$\texttt{Name} \to \texttt{Adresse} \in F^+.$$

Somit ist `Adresse` von `BId` transitiv bezüglich F abhängig.

Beispiel 9.25 (Partielle Abhängigkeit). Wir betrachten ein Schema

$$\mathscr{S}_1 := (\underline{\texttt{Autor}}, \texttt{Jahrgang}, \underline{\texttt{Titel}})$$

mit der Menge

$$F_1 := \{\texttt{Autor} \to \texttt{Jahrgang}\}$$

von funktionalen Abhängigkeiten. Das Schema $\mathscr{S}_1$ modelliert Autoren, deren Jahrgang sowie die Titel der Bücher, die sie geschrieben haben. Wir werden dieses Schema später im Beispiel 9.28 genauer studieren. Hier ist nur wichtig, dass

$$\{\texttt{Autor}, \texttt{Titel}\} \to \{\texttt{Jahrgang}\} \in F_1^+$$

eine partielle Abhängigkeit bezüglich F ist, da $\{\texttt{Autor}\}$ eine echte Teilmenge von $\{\texttt{Autor}, \texttt{Titel}\}$ ist und gilt

$$\{\texttt{Autor}\} \to \{\texttt{Jahrgang}\} \in F_1^+.$$

Es gibt viele verschiedene Normalformen, um gute DB-Schemata zu beschreiben. Wir beschränken uns hier aber auf diejenigen, die in der folgenden Definition zusammengestellt sind.

Definition 9.26 (Normalformen). Wir gehen von einem Relationenschema $\mathscr{S}$ sowie einer Menge F von funktionalen Abhängigkeiten bezüglich $\mathscr{S}$ aus.

Erste Normalform (1NF) $\mathscr{S}$ ist in *erster Normalform*, falls alle Attribute von $\mathscr{S}$ nur atomare Domänen haben. Dabei heisst eine Domäne *atomar*, falls ihre Elemente als nicht-unterteilbare Einheiten aufgefasst werden.

Zweite Normalform (2NF) $\mathscr{S}$ ist in *zweiter Normalform* bezüglich F, falls $\mathscr{S}$ in erster Normalform ist und für alle Attribute A von $\mathscr{S}$ mindestens eine der folgenden zwei Bedingungen erfüllt ist:

(2NF.1) A ist prim;
(2NF.2) A ist nicht von einem Schlüssel für $\mathscr{S}$ partiell bezüglich F abhängig.

Dritte Normalform (3NF) $\mathscr{S}$ ist in *dritter Normalform* bezüglich F, falls $\mathscr{S}$ in erster Normalform ist und für alle $X \to Y$ aus F^+ mindestens eine der folgenden drei Bedingungen erfüllt ist:

(3NF.1) $Y \subseteq X$;
(3NF.2) X ist ein Superschlüssel von $\mathscr{S}$;
(3NF.3) jedes Attribut A aus $Y \setminus X$ ist prim.

Boyce–Codd Normalform (BCNF) $\mathscr{S}$ ist in *Boyce–Codd Normalform* bezüglich F, falls $\mathscr{S}$ in erster Normalform ist und für alle $X \to Y$ aus F^+ mindestens eine der folgenden zwei Bedingungen erfüllt ist:

(BCNF.1) $Y \subseteq X$;
(BCNF.2) X ist ein Superschlüssel von $\mathscr{S}$.

Beispiel 9.27 (Nicht 1NF). Wir wählen das Schema

$$\mathscr{S}_0 := (\underline{\texttt{Autor}}, \texttt{Jahrgang}, \texttt{Titelliste})$$

um Autoren und deren Werke zu verwalten. Das Attribut `Autor` dient als Primärschlüssel. Das heisst, die dazugehörende Menge von funktionalen Abhängigkeiten ist

$$F_0 := \{\texttt{Autor} \to \texttt{Jahrgang}, \texttt{Autor} \to \texttt{Titelliste}\}.$$

Wir betrachten nun folgende Instanz von $\mathscr{S}_0$:

Werke

Autor	**Jahrgang**	**Titelliste**
Goethe	1749	{Götz, Faust}
Schiller	1759	{Tell}

Die Domäne des Attributs `Titelliste` ist hier nicht atomar. Sie besteht aus Mengen, welche aus einzelnen Elementen zusammengesetzt sind. Somit ist dieses Schema *nicht* in 1NF.

Das Problem bei diesem Schema besteht darin, dass nicht auf einen einzelnen Titel zugegriffen werden kann. Das heisst beispielsweise, dass die Query

Wer ist der Autor von `Faust`?

in der relationalen Algebra nicht ausgedrückt werden kann.

Beispiel 9.28 (1NF, aber nicht 2NF). Um das Problem aus dem vorherigen Beispiel zu vermeiden, verwenden wir nun ein Schema bei dem jeder Titel einen eigenen Eintrag erhält. Wir wählen also das Schema

$$\mathscr{S}_1 := (\underline{\texttt{Autor}}, \texttt{Jahrgang}, \underline{\texttt{Titel}}).$$

Die Kombination `Autor`, `Titel` dient als Primärschlüssel. Jedoch ist das Attribut `Jahrgang` natürlich nur vom Attribut `Autor` abhängig. Das heisst, die dazugehörende Menge von funktionalen Abhängigkeiten ist

$$F_1 := \{\texttt{Autor} \rightarrow \texttt{Jahrgang}\}.$$

Damit gilt (siehe auch Beispiel 10.2 später)

$$\{\texttt{Autor}, \texttt{Titel}\} \rightarrow \{\texttt{Autor}, \texttt{Jahrgang}, \texttt{Titel}\} \in F_1^+.$$

Das heisst, die Attributmenge

$$K := \{\texttt{Autor}, \texttt{Titel}\}$$

ist ein Schlüssel für $\mathscr{S}_1$. Wir betrachten nun folgende Instanz von $\mathscr{S}_1$:

Werke

Autor	**Jahrgang**	**Titel**
Goethe	1749	Götz
Goethe	1749	Faust
Schiller	1759	Tell

Alle Attribute haben nun atomare Domänen. Somit ist dieses Schema in 1NF. Jedoch ist es *nicht* in 2NF. Es gilt nämlich:

1. `Jahrgang` ist nicht-prim und
2. `Jahrgang` ist partiell vom Schlüssel K abhängig.

Das Problem hier ist, dass die Daten des Jahrgangs mehrfach vorhanden, d. h. redundant, sind. Dadurch ist es möglich die Integrität der Daten zu verletzen. So könnte der Jahrgang im Tupel von `Götz` verändert werden, ohne dass die entsprechende Änderung im Tupel von `Faust` vorgenommen wird. In diesem Fall wären dann zwei verschiedene (sich widersprechende) Angaben zum Jahrgang von `Goethe` in der Datenbank abgespeichert.

Anmerkung 9.29. Zusammengesetzte Schlüssel wie im vorangehenden Beispiel sind nicht grundsätzlich problematisch. In vielen Fällen sind sie sogar notwendig, beispielsweise um m:n–Beziehungen abzubilden. Es muss nur sichergestellt werden, dass alle nicht-prim Attribute vom *ganzen* Schlüssel abhängig sind. Dies ist im Beispiel 9.28 nicht gegeben.

Beispiel 9.30 (2NF, aber nicht 3NF). Hier betrachten wir nicht mehr ein Schema um Autoren und deren Werke zu verwalten, sondern um einzelne Exemplare dieser Werke zu verwalten. Es soll also möglich sein, das Tupel von `Tell` doppelt einzutragen. Dies wird z. B. von einer Bibliothek benötigt, die von einem Werk mehrere Exemplare besitzen kann.

Wir verwenden dazu das folgende Schema:

$$\mathscr{S}_2 := (\underline{\texttt{BuchId}}, \texttt{Autor}, \texttt{Jahrgang}, \texttt{Titel}).$$

Das neue Attribut `BuchId` dient nun als Primärschlüssel. Wir haben also folgende Menge von funktionalen Abhängigkeiten:

$$F_2 := \big\{\{\texttt{BuchId}\} \to \{\texttt{Autor}, \texttt{Jahrgang}, \texttt{Titel}\}, \{\texttt{Autor}\} \to \{\texttt{Jahrgang}\}\big\}.$$

Wir betrachten nun folgende Instanz von $\mathscr{S}_2$:

Werke

BuchId	**Autor**	**Jahrgang**	**Titel**
1	Goethe	1749	Götz
2	Goethe	1749	Faust
3	Schiller	1759	Tell
4	Schiller	1759	Tell

Das Schema $\mathscr{S}_2$ ist in 2NF. Die Attributmenge {`BuchId`} ist der einzige Schlüssel dieses Schemas und da dieser Schlüssel nicht zusammengesetzt ist (er besteht nur aus einem Attribut), kann kein Attribut partiell von ihm abhängig sein. Somit ist die Bedingung (2NF.2) offensichtlich erfüllt und dieses Schema ist in zweiter Normalform.

Das Schema ist jedoch *nicht* in 3NF. Betrachte die funktionale Abhängigkeit

$$\texttt{Autor} \to \texttt{Jahrgang} \in F_2^+.$$

Wir finden:

1. (3NF.1) ist nicht erfüllt: es gilt nämlich {`Jahrgang`} $\not\subseteq$ {`Autor`}.
2. (3NF.2) ist nicht erfüllt: {`Autor`} ist kein Superschlüssel von $\mathscr{S}_2$.
3. (3NF.3) ist nicht erfüllt: nicht jedes Attribut aus {`Jahrgang`} ist prim, denn das Attribut `Jahrgang` ist nicht Teil eines Schlüssels.

Somit ist keine der drei Bedingungen erfüllt und das Schema $\mathscr{S}_2$ ist nicht in dritter Normalform.

In diesem Schema tritt derselbe Effekt auf wie im vorhergehenden Beispiel. Der Jahrgang eines Autors ist mehrfach abgespeichert, was zu Inkonsistenzen führen kann.

Anmerkung 9.31. Das Problem im obigen Beispiel besteht darin, dass das Attribut `Jahrgang` von `Autor` funktional abhängig ist, aber {`Autor`} kein Superschlüssel ist. Bedingung (3NF.2) wäre erfüllt, falls `Jahrgang` *nur* von Superschlüsseln abhängig ist.

Diese Beobachtung ergibt zusammen mit Bemerkung 9.29 folgende scherzhafte Charakterisierung der dritten Normalform (in Anlehnung an den amerikanischen Gerichtseid): Jedes nicht-prim Attribut muss etwas aussagen über

1. den Schlüssel (1NF),
2. den ganzen Schlüssel (2NF) und
3. nur über den Schlüssel (3NF).

Die Bezugnahme auf *den* Schlüssel ist, wie wir oben gesehen haben, natürlich eine starke Vereinfachung.

Beispiel 9.32 (3NF, aber nicht BCNF). Wir betrachten wiederum das Postleitzahlverzeichnis aus den Beispielen 9.14 und 9.22. Wir haben das Schema

$$\mathscr{S}_3 = (\texttt{Stadt}, \texttt{Str}, \texttt{PLZ})$$

mit der dazugehörigen Menge

$$F_3 := \big\{ \{\texttt{Stadt}, \texttt{Str}\} \to \{\texttt{PLZ}\}, \quad \{\texttt{PLZ}\} \to \{\texttt{Stadt}\} \big\}$$

von funktionalen Abhängigkeiten. Schlüssel von $\mathscr{S}_3$ sind also die Attributmengen $\{\texttt{Stadt}, \texttt{Str}\}$ und $\{\texttt{Str}, \texttt{PLZ}\}$.

Betrachte folgende Instanz von $\mathscr{S}_3$

Verzeichnis

PLZ	Str	Stadt
2500	Baumstr	Biel
3000	Parkstr	Bern
3018	Wiesenstr	Bern
3018	Baumstr	Bern

Dieses Schema ist in 3NF. In der Tat gehört jedes Attribut zu einem Schlüssel, d. h. alle Attribute sind prim. Somit ist die Bedingung (3NF.3) für alle funktionalen Abhängigkeiten aus F_3^+ erfüllt und damit ist $\mathscr{S}_3$ in dritter Normalform.

Das Schema ist jedoch *nicht* in BCNF. Betrachte die funktionale Abhängigkeit

Wir finden: $$\texttt{PLZ} \to \texttt{Stadt} \in F_3^+.$$

1. (BCNF.1) ist nicht erfüllt: es gilt nämlich $\{\texttt{Stadt}\} \not\subseteq \{\texttt{PLZ}\}$.
2. (BCNF.2) ist nicht erfüllt: $\{\texttt{PLZ}\}$ ist kein Superschlüssel von $\mathscr{S}_3$.

Somit sind beide Bedingungen nicht erfüllt und das Schema $\mathscr{S}_3$ ist nicht in Boyce–Codd Normalform.

Diese Verletzung der BCNF bedeutet, dass immer noch gewisse Redundanzen vorhanden sein können. Betrachten wir die Relation `Verzeichnis`. Dort ist die Beziehung zwischen der Postleitzahl 3018 und dem Ortsnamen `Bern` mehrfach abgespeichert. Sollte sich die Post entschliessen die Ortsbezeichnung für diese Postleitzahl zu ändern, beispielsweise zu `Bern-Bümpliz`, so sind Updates in mehreren Tupeln nötig. Falls nicht alle diese Updates ausgeführt werden, dann sind verschiedene Namen für dieselbe Postleitzahl abgespeichert. Wird der Eintrag der `Baumstrasse` aktualisiert, derjenige der `Wiesenstrasse` aber nicht, so ist die funktionale Abhängigkeit

$$\texttt{PLZ} \to \texttt{Stadt}$$

nicht mehr erfüllt. Die Relation ist dann inkonsistent.

Anmerkung 9.33. Wie das obige Beispiel zeigt, können Updates auf einem Schema in 3NF noch zu Inkonsistenzen führen. Ist jedoch ein Schema in BCNF bezüglich einer Menge F von funktionalen Abhängigkeiten, so kann es keine Redundanzen geben, welche durch F verursacht werden. Das heisst, in einem Schema, welches in BCNF ist, können Updates nicht zu Inkonsistenzen bezüglich funktionaler Abhängigkeiten führen.

Die Frage lautet somit:

Gibt es zu jedem DB-Schema ein äquivalentes DB-Schema in BCNF?

Wir haben folgendes Theorem.

Theorem 9.34. *Gegeben seien ein Schema $\mathscr{S}$ und eine Menge von funktionalen Abhängigkeiten F bezüglich $\mathscr{S}$. Dann gilt:*

1. *Es gibt eine verlustfreie Zerlegung*

$$\mathscr{Z} := \{\mathscr{S}_1, \ldots, \mathscr{S}_n\}$$

   von $\mathscr{S}$, so dass alle $\mathscr{S}_i \in \mathscr{Z}$ in BCNF bezüglich $\Pi_{\mathscr{S}_i}(F)$ sind.
2. *Es gibt eine verlustfreie und abhängigkeitserhaltende Zerlegung*

$$\mathscr{Z} := \{\mathscr{S}_1, \ldots, \mathscr{S}_n\}$$

   von $\mathscr{S}$, so dass alle $\mathscr{S}_i \in \mathscr{Z}$ in 3NF bezüglich $\Pi_{\mathscr{S}_i}(F)$ sind.

Damit gibt es Zerlegungen in BCNF und 3NF. Jedoch ist nur die Zerlegung in 3NF abhängigkeitserhaltend. Bei einer Zerlegung in BCNF können funktionale Abhängigkeiten verloren gehen.

Wir verzichten hier auf den Beweis dieses Theorems und geben nur zwei Beispiele an, wie eine Zerlegung aussehen kann. Im nächsten Kapitel werden wir dann Algorithmen studieren, um ein gegebenes Schema in 3NF und BCNF zu zerlegen.

Beispiel 9.35 (Zerlegung). Wir betrachten nochmals die Situation aus Beispiel 9.28. Wir haben das Schema

$$\mathscr{S}_1 := (\underline{\texttt{Autor}}, \texttt{Jahrgang}, \underline{\texttt{Titel}})$$

mit den funktionalen Abhängigkeiten

$$F_1 := \{\texttt{Autor} \to \texttt{Jahrgang}\}.$$

Dieses Schema verletzt die zweite Normalform.

Wir zerlegen $\mathscr{S}_1$ in $\{\mathscr{S}_{1,1}, \mathscr{S}_{1,2}\}$ mit

$$\mathscr{S}_{1,1} := \{\texttt{Autor}, \texttt{Jahrgang}\} \text{ und } \mathscr{S}_{1,2} := \{\texttt{Autor}, \texttt{Titel}\}.$$

Mit Lemma 9.19 finden wir, dass diese Zerlegung verlustfrei ist. Zusätzlich ist sie auch noch abhängigkeitserhaltend. Insbesondere haben wir

$$\{\texttt{Autor} \to \texttt{Jahrgang}\} \in \Pi_{\mathscr{S}_{1,1}}(F).$$

Weiter stellen wir fest:

1. $\mathscr{S}_{1,1}$ ist in BCNF bezüglich $\Pi_{\mathscr{S}_{1,1}}(F)$,
2. $\mathscr{S}_{1,2}$ ist in BCNF bezüglich $\Pi_{\mathscr{S}_{1,2}}(F)$.

Die Relation `Werke` aus Beispiel 9.28 wird wie folgt zerlegt:

$\pi_{S_{1,1}}$ **(Werke)**

Autor	**Jahrgang**
Goethe	1749
Schiller	1759

$\pi_{S_{1,2}}$ **(Werke)**

Autor	**Titel**
Goethe	Götz
Goethe	Faust
Schiller	Tell

Hier haben wir eine Zerlegung in BCNF gesehen, welche abhängigkeitserhaltend ist. Im Allgemeinen muss dies nicht erfüllt sein, wie das folgende Beispiel zeigt.

Beispiel 9.36 (Zerlegung mit Abhängigkeitsverlust). Wir betrachten die Situation aus Beispiel 9.32. Das heisst,

$$\mathscr{S}_3 = (\texttt{Stadt}, \texttt{Str}, \texttt{PLZ})$$

und

$$F_3 := \big\{ \{\texttt{Stadt}, \texttt{Str}\} \to \{\texttt{PLZ}\}, \quad \{\texttt{PLZ}\} \to \{\texttt{Stadt}\} \big\}.$$

Wir hatten gezeigt, dass $\mathscr{S}_3$ nicht in BCNF bezüglich F_3 ist.

Wir wählen nun die Zerlegung $\{\mathscr{S}_{3,1}, \mathscr{S}_{3,2}\}$ mit

$$\mathscr{S}_{3,1} := \{\texttt{Str}, \texttt{PLZ}\} \text{ und } \mathscr{S}_{3,2} := \{\texttt{Stadt}, \texttt{PLZ}\}.$$

Im Beispiel 9.22 haben wir gesehen, dass diese Zerlegung nicht abhängigkeitserhaltend ist. Aus Lemma 9.19 folgt jedoch, dass sie verlustfrei ist. Weiter stellen wir fest:

1. $\mathscr{S}_{3,1}$ ist in BCNF bezüglich $\Pi_{\mathscr{S}_{3,1}}(F)$,
2. $\mathscr{S}_{3,2}$ in ist BCNF bezüglich $\Pi_{\mathscr{S}_{3,2}}(F)$.

Die Relation `Verzeichnis` aus Beispiel 9.32 wird also wie folgt zerlegt:

$\pi_{\mathscr{S}_{3,1}}$**(Verzeichnis)**

Str	**PLZ**
Baumstr	2500
Parkstr	3000
Wiesenstr	3018
Baumstr	3018

$\pi_{\mathscr{S}_{3,2}}$**(Verzeichnis)**

Stadt	**PLZ**
Biel	2500
Bern	3000
Bern	3018

In den folgenden Lemmata fassen wir einige Eigenschaften der eingeführten Normalformen zusammen.

Lemma 9.37. *Gegeben seien ein Relationenschema $\mathscr{S}$ sowie eine Menge F von funktionalen Abhängigkeiten bezüglich $\mathscr{S}$. Dann ist $\mathscr{S}$ in 3NF bezüglich F genau dann, wenn es kein nicht-primes Attribut A von $\mathscr{S}$ gibt, das von einem Schlüssel für $\mathscr{S}$ transitiv bezüglich F abhängig ist.*

Beweis. Wir zeigen zuerst die Richtung von links nach rechts und nehmen dazu an, dass $\mathscr{S}$ in 3NF bezüglich F ist. Nun gehen wir indirekt vor und nehmen zusätzlich an, dass A ein nicht-primes Attribut ist, das transitiv bezüglich F von einem Schlüssel X für $\mathscr{S}$ abhängt. Dann gibt es ein Y mit $A \notin X \cup Y$, so dass

$$X \to Y \in F^+, \quad Y \to X \notin F^+ \quad \text{und} \quad Y \to A \in F^+ \tag{9.5}$$

gilt. Nun wissen wir, dass für $Y \to A$ eine der drei Bedingungen (3NF.1), (3NF.2) oder (3NF.3) erfüllt ist. Wegen $A \notin X \cup Y$ und da A nicht-prim ist, muss es sich also um (3NF.2) handeln. Folglich ist Y ein Superschlüssel für $\mathscr{S}$. Daraus folgt aber $Y \to X \in F^+$. Dies ist ein Widerspruch zu (9.5), so dass die Richtung von links nach rechts nachgewiesen ist.

Zum Beweis der Richtung von rechts nach links gehen wir davon aus, dass es kein nicht-primes Attribut von $\mathscr{S}$ gibt, das von einem Schlüssel für $\mathscr{S}$ transitiv bezüglich F abhängig ist. Ausserdem wählen wir eine funktionale Abhängigkeit $X \to Y \in F^+$, für die

$$Y \not\subseteq X \quad \text{und} \quad X \text{ ist kein Superschlüssel für } \mathscr{S}$$

vorausgesetzt wird. Ferner betrachten wir ein Attribut $A \in Y \setminus X$. Wegen

$$X \to Y \in F^+$$

gilt auch

$$X \to A \in F^+. \tag{9.6}$$

Da X kein Superschlüssel für $\mathscr{S}$ ist, gibt es einen Schlüssel Z für $\mathscr{S}$ mit der Eigenschaft

$$Z \to X \in F^+ \quad \text{und} \quad X \to Z \notin F^+. \tag{9.7}$$

Mit (9.6) und (9.7) folgt also, dass A von einem Schlüssel für $\mathscr{S}$, nämlich Z, transitiv bezüglich F abhängig ist. Daher ist A prim und es folgt (3NF.3). Damit ist unser Beweis vollständig. □

Beispiel 9.38. Wir gehen zurück zum Beispiel 9.24. Wir haben also das Schema

$$\mathscr{A} := (\texttt{BId}, \texttt{Name}, \texttt{Adresse}, \texttt{Datum})$$

mit den funktionalen Abhängigkeiten

$$F := \big\{\{\texttt{BId}\} \to \{\texttt{Name}, \texttt{Adresse}, \texttt{Datum}\}, \{\texttt{Name}\} \to \{\texttt{Adresse}\}\big\}.$$

Wir wissen:

1. `Adresse` ist ein nicht-primes Attribut,
2. `BId` ist ein Schlüssel für $\mathscr{A}$,
3. `Adresse` ist transitiv abhängig von `BId`.

Mit Lemma 9.37 folgt somit, dass $\mathscr{A}$ nicht in 3NF ist.

Umgekehrt folgt aus Lemma 9.37 auch, dass es in jedem Schema das nicht in 3NF ist, transitive Abhängigkeiten geben muss. Somit werden in jedem Schema das nicht in 3NF ist, dieselben Anomalien auftreten, die wir auch für $\mathscr{A}$ im Abschn. 9.1 beschrieben haben.

Lemma 9.39. *Es gilt folgende Beziehung:*

$$\text{BCNF} \implies \text{3NF} \implies \text{2NF} \implies \text{1NF}.$$

Beweis. Offensichtlich ist nur zu zeigen, dass aus der dritten Normalform die zweite Normalform folgt. Sei also $\mathscr{S}$ ein Relationenschema in 3NF bezüglich F, und sei A ein Attribut von $\mathscr{S}$. Ist A prim, so ist Bedingung (2NF.1) erfüllt, und wir sind fertig.

Ist andererseits A nicht-prim, so folgt nach Lemma 9.37, dass

$$A \text{ kann nicht von einem Schlüssel für } \mathscr{S} \text{ transitiv bezüglich } F \text{ abhängig sein.} \tag{9.8}$$

Nun gehen wir indirekt vor und nehmen an, dass

$$\text{(2NF.2) nicht erfüllt ist.} \tag{9.9}$$

Das heisst, A ist von einem Schlüssel X für $\mathscr{S}$ partiell bezüglich F abhängig. Dann gibt es eine echte Teilmenge Z von X, so dass

$$Z \to A \in F^+ \tag{9.10}$$

gilt. Da A nicht-prim ist, kann A kein Element von X sein. Also haben wir

$$A \notin X \quad \text{und} \quad A \notin Z. \tag{9.11}$$

Da X ein Schlüssel für $\mathscr{S}$ und Z eine echte Teilmenge von X ist, dürfen wir ferner schliessen auf

$$X \to Z \in F^+ \quad \text{und} \quad Z \to X \notin F^+. \tag{9.12}$$

Aus (9.10), (9.11) und (9.12) folgt schliesslich, dass A vom Schlüssel X transitiv bezüglich F abhängig ist. Dies ist jedoch ein Widerspruch zu (9.8). Damit ist (9.9) nicht erfüllbar und Bedingung (2NF.2) muss gelten. □

Weiterführende Literatur[2]

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases: The Logical Level. Addison-Wesley, Reading (1995)
2. Codd, E.F.: Further normalization of the data base relational model. IBM Research Report, San Jose, **RJ909** (1971)
3. Codd, E.F.: Relational completeness of data base sublanguages. In: Rustin, R. (Hrsg.) Courant Computer Science Symposium 6: Data Base Systems, S. 33–64. Prentice Hall, Englewood Cliffs (1972)
4. Codd, E.F.: Recent investigations in relational data base systems. In: IFIP Congress, S. 1017–1021 (1974)
5. Jäger, G.: Datenbanken. Vorlesungsskript Universität Bern, Bern (1999)
6. Kandzia, P., Klein, H.: Theoretische Grundlagen relationaler Datenbanksysteme „Reihe Informatik“, Bd. 79. Bibliographisches Institut (1993)
7. Kent, W.: A simple guide to five normal forms in relational database theory. Commun. ACM **26**(2), 120–125 (1983)
8. Maier, D.: The Theory of Relational Databases. Computer Science Press (1983). http://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html. Zugegriffen am 11.06.2019

[2]Codd entwickelte die Konzepte von funktionalen Abhängigkeiten und Normalformen (insbesondere auch die dritte Normalform) bereits in seinen ersten Arbeiten zum relationalen Modell [2, 3]. Die Boyce–Codd Normalform geht ebenfalls auf die frühen Arbeiten zurück [4]. Unsere Präsentation dieser Themen basiert auf dem Vorlesungsskript von Jäger [5]. Das Buch von Meier [8], welches frei verfügbar ist, bietet ebenfalls eine hervorragende Darstellung dieses Materials. Normalformen und Abhängigkeiten werden natürlich auch in den Theorie-Büchern von Abiteboul et al. [1] und von Kandzia und Klein [6] ausführlich besprochen. Die Charakterisierung der dritten Normalform in Bemerkung 9.31 stammt aus [7].