Das Relationenmodell

2

Die Grundidee des Relationenmodells ist es, Daten in Form von Relationen abzuspeichern. In diesem Kapitel führen wir die wesentlichen Begriffe des Relationenmodells ein, wie beispielsweise Attribut, Domäne, Schema und Instanz. Ausserdem diskutieren wir das Konzept des Primärschlüssels eines Schemas. Im letzten Abschnitt behandeln wir Integritätsbedingungen auf einem Datenbankschema. Insbesondere definieren wir die Bedeutung von Fremdschlüsseln, unique Constraints und not null Constraints.

2.1 Struktur relationaler Datenbanken

Das mathematische Konzept der Relation bildet das theoretische Grundgerüst des Relationenmodells. Dieses ermöglicht eine formale Beschreibung der Struktur relationaler Datenbanken. Damit lassen sich Fragen im Zusammenhang mit Semantik, Integrität und Redundanz präzise untersuchen.

Im Relationenmodell sind die Daten, welche in der Datenbank gehalten werden, in der Form von Relationen abgespeichert. Als erstes beobachten wir, dass n-stellige Relationen als Tabellen mit n Spalten dargestellt werden können.

Beispiel 2.1. Wir nehmen an, wir wollen die Daten einer Menge von Autos in unserer Datenbank halten. Für jedes Auto soll dessen Marke und Farbe abgespeichert werden. Wir können also jedes Auto als Paar (`Marke`, `Farbe`) darstellen. Eine Menge von Autos entspricht somit einer Menge von solchen Paaren. Das heisst wir können eine Menge von Autos durch eine 2-stellige Relation repräsentieren. Wir betrachten nun folgendes konkretes Beispiel:

```
Autos := {(Opel,silber),(VW,rot),(Audi,schwarz)}.
```


T. Studer, *Relationale Datenbanken*,
https://doi.org/10.1007/978-3-662-58976-2_2

Die Relation Autos enthält in diesem Beispiel die Daten zu drei Autos.

Wir können diese Relation als Tabelle mit zwei Spalten darstellen:

Opel	silber
VW	rot
Audi	schwarz

Wir werden nun die Grundbegriffe des Relationenmodells einführen. Diese erlauben uns präzise über dieses Modell zu sprechen und formale Definitionen der Datenbank-Operationen auf Relationen anzugeben.

Attribut und Domäne

Wir geben jeder Spalte der Tabellendarstellung einer Relation einen Namen. Diese Namen heissen *Attribute*. Im obigen Beispiel nennen wir die linke Spalte Marke und die rechte Spalte nennen wir Farbe. Die Attribute sind also Namen für Eigenschaften von Objekten. In unserem Beispiel sind Marke und Farbe Eigenschaften von Autos.

In der Tabellenform geben wir den Namen der Relation und die Attribute als Überschriften folgendermassen an:

Autos	
Marke	**Farbe**
Opel	silber
VW	rot
Audi	schwarz

Die Menge von möglichen Werten, die ein Attribut annehmen kann, bezeichnen wir als die *Domäne* dieses Attributs.

Es kann vorkommen, dass der Wert eines Attributes unbekannt ist. Dies kann sein, weil wir den Wert des Attributs nicht kennen oder weil es noch keinen Wert besitzt. Nehmen wir an, die Relation Autos hat noch ein drittes Attribut Fahrer. Es kann sein, dass wir den Fahrer eines Autos nicht kennen oder dass es noch keinen Fahrer hat (bspw. weil es noch nicht verkauft wurde). Um diese Fälle zu modellieren, führen wir einen speziellen Wert Null ein, *welcher zu jeder Domäne gehört*. Wir verwenden Null, um auszudrücken, dass der Wert eines Attributs unbekannt ist. Wir werden also dem Attribut Fahrer den Wert Null zuweisen, falls wir den Fahrer nicht kennen oder falls das Auto noch keinen Fahrer hat.

In der Tabellenform schreiben wir häufig − anstelle von Null. Nehmen wir also an, der Opel wird von Tom und der Audi von Eva gefahren. Der VW sei noch nicht verkauft. Wir können diese Information wie folgt darstellen:

Autos

Marke	Farbe	Fahrer
Opel	silber	Tom
VW	rot	-
Audi	schwarz	Eva

Da wir `Null` verwenden, um auszudrücken, dass der Wert eines Attributs unbekannt ist, erhält `Null` eine spezielle Semantik bezüglich der Gleichheit:

$$\text{Es gilt } \textit{nicht}, \text{ dass } \texttt{Null} = \texttt{Null}. \tag{2.1}$$

Wenn wir zwei unbekannte Werte vergleichen, so wissen wir eben nicht, ob sie gleich sind. Deshalb verwenden wir eine Semantik für die (2.1) der Fall ist.[1]

An einigen Stellen werden wir jedoch zwei `Null` Werte als gleichwertig betrachten müssen. Dazu führen wir folgende schwache Gleichheitsrelation zwischen zwei atomaren Objekten a und b ein:

$$a \simeq b \qquad \text{g.d.w.} \qquad a = b \quad \text{oder} \quad (\text{a ist } \texttt{Null} \text{ und b ist } \texttt{Null}).$$

Für zwei n-Tupel definieren wir analog:

$$(a_1, \ldots, a_n) \simeq (b_1, \ldots, b_n) \qquad \text{g.d.w.} \qquad a_i \simeq b_i \text{ für alle } 1 \leq i \leq n.$$

Relation und n-Tupel über Domänen $D_1, \ldots, D_n$

Gegeben seien n Domänen $D_1, \ldots, D_n$. Ein *n-Tupel über* $D_1, \ldots, D_n$ ist ein Objekt der Form

$$(a_1, \ldots, a_n),$$

wobei $a_i \in D_i$ für alle $1 \leq i \leq n$. Eine n-stellige *Relation* R über den Domänen $D_1, \ldots, D_n$ ist eine Menge von n-Tupeln über $D_1, \ldots, D_n$. Das heisst

$$R \subseteq \{(x_1, \ldots, x_n) \mid x_1 \in D_1 \text{ und } \cdots \text{ und } x_n \in D_n\}.$$

[1] In der Beschreibung der Selektionsoperation (Seite 48) werden wir im Detail auf die Semantik des `Null` Wertes eingehen.

Relationenschema

Relationenschemata (oder einfach nur *Schemata*) spezifizieren die bei Relationen verwendeten Attribute und Domänen. Es handelt sich dabei um Sequenzen der Form

$$(A_1 : D_1, \ldots, A_n : D_n),$$

wobei $A_1, \ldots, A_n$ Attribute mit den jeweiligen Domänen $D_1, \ldots, D_n$ sind. Ergeben sich die Domänen unmittelbar aus dem Kontext oder sind sie unwichtig, so schreiben wir manchmal nur

$$(A_1, \ldots, A_n)$$

anstelle von $(A_1 : D_1, \ldots, A_n : D_n)$. Weiter verwenden wir die Sprechweise *R ist eine Relation über $A_1, \ldots, A_n$* und meinen damit, dass R eine Relation über den dazugehörenden Domänen $D_1, \ldots, D_n$ ist. Dafür sagen wir auch *R ist eine Instanz des Schemas* $(A_1, \ldots, A_n)$.

Relationales Datenbank-Schema

Als *relationales Datenbank-Schema* (oder kurz *DB-Schema*) bezeichnen wir die Menge aller verwendeten Relationenschemata.

Relationale Datenbank

Als *relationale Datenbank* (oder kurz *relationale DB*) bezeichnen wir das verwendete relationale Datenbank-Schema zusammen mit den momentanen Werten der Relationen. Eine relationale Datenbank besteht somit aus einem DB-Schema zusammen mit den aktuellen Instanzen aller Schemata des DB-Schemas. Wir sprechen in diesem Zusammenhang auch von einer *Instanz* eines DB-Schemas und meinen damit die Menge der aktuellen Instanzen aller Schemata des DB-Schemas.

2.2 Schlüssel

Beim Entwurf einer Datenbank ist es wichtig festzulegen, wie verschiedene Objekte auf sinnvolle Weise unterschieden werden können. In der realen Welt handelt es sich dabei tatsächlich um unterschiedliche Objekte. Im relationalen Modell müssen wir explizit angeben, mit Hilfe von welchen Attributen wir diese unterschiedlichen Objekte unterscheiden können. Das heisst, wir müssen angeben, welche Attribute wir zur Identifikation von Objekten verwenden wollen.

Dazu führen wir folgende Notationen ein. Es seien $A_1, \ldots, A_n$ Attribute,

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

ein Relationenschema,[2] und R eine Instanz von $\mathscr{S}$.

1. Ist t ein n-Tupel, das zu R gehört, so schreiben wir $t[A_i]$ für den Wert von t bei Attribut A_i. Für $(a_1, \ldots, a_i, \ldots, a_n) \in R$ heisst das

$$(a_1, \ldots, a_i, \ldots, a_n)[A_i] = a_i.$$

2. Ist $K = (A_{i_1}, \ldots, A_{i_m})$ eine Sequenz von Attributen, so definieren wir für ein n-Tupel $(a_1, \ldots, a_i, \ldots, a_n) \in R$

$$(a_1, \ldots, a_i, \ldots, a_n)[K] := (a_{i_1}, \ldots, a_{i_m}). \tag{2.2}$$

   Für $s, t \in R$ bedeutet also $s[K] = t[K]$, dass die Werte von s und t in allen Attributen aus K übereinstimmen.

Die Frage in diesem Abschnitt lautet ja, wie wir in einer Instanz R von $\mathscr{S}$ die einzelnen Elemente unterscheiden können. Dazu wählen wir einen sogenannten *Primärschlüssel*. Dies ist eine Sequenz von Attributen

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

Dann verlangen wir für alle Instanzen R von $\mathscr{S}$ und alle $s, t \in R$, dass

$$s[K] = t[K] \quad \Longrightarrow \quad s \simeq t. \tag{2.3}$$

Das heisst, wir können jedes Element t aus R eindeutig identifizieren anhand der Werte von t bei den Primärschlüssel-Attributen. Wenn die Werte von zwei Elementen s und t bei den Primärschlüssel-Attributen übereinstimmen, so müssen s und t identisch sein.

Zu jedem Relationenschema können wir nur *einen* Primärschlüssel wählen. Wir geben diesen an, indem wir beim Schema diejenigen Attribute unterstreichen, welche zum gewählten Primärschlüssel gehören.

Beispiel 2.2. Betrachten wir eine Relation `Autos` über den Attributen `Marke`, `Farbe`, `Baujahr`, `Vorname` und `Nachname`, welche neben Marke und Farbe der Autos auch das Baujahr, sowie Vor- und Nachname des Fahrers enthält.

[2] Das Zeichen $\mathscr{S}$ ist ein kaligraphisches S.

Autos

Marke	Farbe	Baujahr	Vorname	Nachname
Opel	silber	2010	Tom	Studer
Opel	schwarz	2010	Eva	Studer
VW	rot	2014	Eva	Studer
Audi	schwarz	2014	Eva	Meier

Es sei nun

$$t := (\texttt{Opel, schwarz, 2010, Eva, Studer}).$$

Damit gilt

$$t[(\texttt{Marke, Farbe})] = (\texttt{Opel, schwarz})$$

und

$$t[(\texttt{Nachname, Baujahr})] = (\texttt{Studer, 2010}).$$

Für die gegebene Relation Autos sind unter anderem die beiden folgenden Primärschlüssel möglich:

$$(\texttt{Marke}, \texttt{Farbe}) \quad \text{und} \quad (\texttt{Baujahr}, \texttt{Vorname}, \texttt{Nachname}). \tag{2.4}$$

Falls wir (Marke, Farbe) als Primärschlüssel wählen, so geben wir das Schema wie folgt an:

$$(\underline{\texttt{Marke}}, \underline{\texttt{Farbe}}, \texttt{Baujahr}, \texttt{Vorname}, \texttt{Nachname}).$$

Gemäss der Definition ist ein Primärschlüssel eine Sequenz von Attributen, d. h. die Attribute sind im Primärschlüssel geordnet. Diese Ordnung wird in der obigen Notation natürlich nicht angezeigt. Dies muss uns aber im Moment nicht stören.

In einer echten Datenbankanwendung sind wahrscheinlich beide möglichen Primärschlüssel aus (2.4) ungeeignet. Es ist nämlich gut möglich, dass wir später dieser Relation weitere Autos hinzufügen möchten. Da kann es dann sein, dass ein zweiter roter VW eingefügt werden soll oder ein weiteres Auto mit Baujahr 2010 und dem Fahrer Tom Studer.

Später, in Definition 9.13 werden wir noch im Detail darauf eingehen, welche Eigenschaften ein Primärschlüssel haben sollte und welche Attributmengen sich gut als Primärschlüssel eignen.

In der Praxis wird oft ein zusätzliches Attribut, nennen wir es `Auto_Id`, hinzugefügt, welches als Primärschlüssel dient. Dieses Attribut hat nur den Zweck, die verschiedenen Elemente der Relation eindeutig zu bestimmen. Es beschreibt aber keine echte Eigenschaft von Autos. Wir nennen einen solchen Primärschlüssel einen *nicht-sprechenden* Schlüssel. Ein *sprechender* Schlüssel hingegen hat eine logische Beziehung zu einem oder mehreren Attributen des Schemas.

Anmerkung 2.3. Es ist gute Praxis *keine* sprechenden Schlüssel zu verwenden, da diese die Tendenz haben zu zerbrechen. Das heisst, früher oder später wird eine Situation auftreten, in der ein neues Tupel eingefügt werden soll, dessen sprechender Schlüssel bereits ein Tupel in der Relation bezeichnet. Oder es kann sein, dass das System der Schlüsselgenerierung komplett geändert wird. Beispiele dazu sind:

1. das System der AHV-Nummern (eindeutige Personennummer der Alters- und Hinterlassenenversicherung) in der Schweiz, welches 2008 geändert wurde,
2. die Internationale Standardbuchnummer (ISBN), für die 2005 ein revidierter Standard eingeführt wurde.

Beispiel 2.4. In (2.3) verwenden wir auf der rechten Seite der Implikation die schwache Gleichheit. Hier zeigen wir weshalb wir diese Definition gewählt haben. Dazu betrachten wir die Tabelle

Autos

Marke	**Farbe**	**Baujahr**
Opel	silber	Null

mit dem Primärschlüssel $K = (\texttt{Marke}, \texttt{Farbe})$. Wir setzen

$$s := t := (\texttt{Opel}, \texttt{Silber}, \texttt{Null}) .$$

Es gilt $s[K] = t[K]$. Um (2.3) zu erfüllen, muss nun auch $s \simeq t$ gelten, was offensichtlich der Fall ist. Jedoch gilt *nicht* $s = t$, weil $s[\texttt{Baujahr}] = t[\texttt{Baujahr}]$ nicht gilt. Somit würde diese Relation eine Version von (2.3) mit der Bedingung $s = t$ nicht erfüllen.

Betrachten wir noch einmal die Tabelle aus Beispiel 2.2. In dieser Tabelle sind die Daten zu Eva Studer doppelt abgespeichert. Dies kann zu einer Reihe von Problemen führen: z. B. falls sie den Namen ändert, so muss diese Änderung an zwei Stellen in der Datenbank nachgeführt werden. Dies birgt die Gefahr von inkonsistenten Daten. Es könnte nämlich sein, dass nur eine Änderung ausgeführt wird, die andere aber nicht und somit diese Fahrerin mit zwei verschiedenen Namen in der Datenbank vorhanden ist.

Besser ist es, die Daten zu den Autos und zu den Fahrern in zwei separaten Tabellen zu speichern. So ist es möglich, dass die Angaben zu jeder Person nur einmal abgespeichert

sind. Damit können Inkonsistenzen ausgeschlossen werden. Das folgende Beispiel illustriert dieses Vorgehen.

Beispiel 2.5. Die Daten zu Autos und deren Fahrer aufgeteilt in zwei Tabellen. Die Primärschlüssel sind jeweils unterstrichen.

Autos

Marke	**Farbe**	**Baujahr**	**FahrerId**
Opel	silber	2010	1
Opel	schwarz	2010	2
VW	rot	2014	2
Audi	schwarz	2014	3

Personen

PersId	**Vorname**	**Nachname**
1	Tom	Studer
2	Eva	Studer
3	Eva	Meier

Die Verknüpfung zwischen den beiden Tabellen geschieht über ein neues Attribut `FahrerId` in der Tabelle `Autos`, welches jedem Auto einen Fahrer zuordnet. Dazu enthält `FahrerId` den Primärschlüssel (d. h. den Wert von `PersId`) des entsprechenden Fahrers.

In dieser Situation sagen wir, `FahrerId` *referenziert* die Tabelle `Personen` oder `FahrerId` ist ein *Fremdschlüssel* für `Personen`.

Wenn wir also wissen wollen, wer den roten VW fährt, so suchen wir in der Tabelle `Autos` das Tupel für den roten VW. Dieses ist eindeutig, da (`Marke`, `Farbe`) der Primärschlüssel dieser Relation ist. Wir nehmen nun den Wert des `FahrerId` Attributs dieses Tupels, hier also den Wert 2. Da `FahrerId` ein Fremdschlüssel für `Personen` ist, wissen wir, dass es genau einen Eintrag in der Tabelle `Personen` gibt, dessen Primärschlüssel den Wert 2 hat. Wir suchen diesen Eintrag und erfahren so, dass der rote VW von Eva Studer gefahren wird.

2.3 Integritätsbedingungen

Unter *Integritätsbedingungen* (oder *Constraints*) werden Zusicherungen verstanden, welche die in der Datenbank enthaltenen Daten betreffen. Damit kann die Menge der erlaubten Instanzen eines Datenbankschemas eingeschränkt werden. Wir unterscheiden zwei Arten von Integritätsbedingungen:

1. *Strukturelle Regeln* (statische Integritätsbedingungen), die in einem Zustand erfüllt sein müssen, damit er erlaubt ist.
2. *Verhaltensregeln* (dynamische Integritätsbedingungen), die bei der Ausführung von Änderungen erfüllt sein müssen.

Strukturelle Regeln betreffen unter anderem die Wahl von Primär- und Fremdschlüsseln im Relationenmodell. Eine strukturelle Regel kann auch verlangen, dass gewisse Attribute nicht den Wert `Null` annehmen dürfen. Zum Beispiel könnte eine statische Integritätsbedingung ausdrücken, dass jedes Auto ein Baujahr haben muss, d. h., dass das Attribut `Baujahr` in der Relation `Autos` nicht `Null` sein darf.

Verhaltensregeln werden an Zustandsänderungen gestellt. Sie bestimmen welche Veränderungen der gespeicherten Daten erlaubt sind und welche nicht. Eine dynamische Integritätsbedingung könnte z. B. verlangen, dass der Vorname einer Person nicht geändert werden darf.

Im Folgenden geben wir eine Reihe von statischen Integritätsbedingungen an, welche wir von einer Datenbank verlangen können.

Unique Constraints

Es seien $A_1, \ldots, A_n$ Attribute. Ein *unique Constraint* auf einem Schema

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

ist bestimmt durch eine Sequenz von Attributen

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

Definition 2.6 (Integritätsregel: unique Constraint). Gegeben sei ein Relationenschema $\mathscr{S}$ mit einem unique Constraint U. Für jede Instanz R von $\mathscr{S}$ und alle $s, t \in R$ muss gelten

$$s[U] = t[U] \quad \Longrightarrow \quad s \simeq t.$$

Ein unique Constraint U auf einem Schema $\mathscr{S}$ besagt, dass jede Kombination von Werten der Attribute aus U nur einmal vorkommen darf. Für eine Instanz R von $\mathscr{S}$ kann somit jedes Tupel $t \in R$ eindeutig anhand der Werte in den Attributen aus U bestimmt werden. Dies gilt jedoch nur, falls die Werte in U nicht `Null` sind.

Beispiel 2.7. Betrachte das folgende Schema für Autos

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

mit einem unique Constraint

$$U := \texttt{(Marke,Farbe)}.$$

Die folgende Instanz `Autos` von $\mathscr{S}$ erfüllt den unique Constraint U:

Autos

Marke	Farbe	Baujahr
Opel	silber	2010
Null	schwarz	2012
Null	schwarz	2014

In der Tat ist es *nicht* der Fall, dass

$$\texttt{(Null,schwarz) = (Null,schwarz)},$$

da `Null` $=$ `Null` nicht gilt. Somit ist die Integritätsregel 1 erfüllt, obwohl die beiden Einträge mit (`Null`, `schwarz`) unterschiedliche Werte für das Attribut `Baujahr` haben.

Not null Constraints

Es seien $A_1, \ldots, A_n$ Attribute. Ein *not null Constraint* auf einem gegebenen Schema $\mathscr{S} = (A_1, \ldots, A_n)$ ist bestimmt durch ein Attribut A_i.

Definition 2.8 (Integritätsregel: not null Constraint). Gegeben sei ein Relationenschema $\mathscr{S}$ mit einem not null Constraint A_i. Für jede Instanz R von $\mathscr{S}$ und alle $s \in R$ muss gelten

$$s[A_i] \text{ hat nicht den Wert } \texttt{Null}.$$

Primary key Constraints

Ein primary key Constraint wird durch einen Primärschlüssel definiert. Er setzt sich zusammen aus einem unique Constraint für den Primärschlüssel und not null Constraints für alle Attribute des Primärschlüssels.

Definition 2.9 (Integritätsregel: primary key Constraint). Gegeben sei ein Relationenschema $\mathscr{S}$ mit einem Primärschlüssel K. Für jede Instanz R von $\mathscr{S}$ und alle $s, t \in R$ muss gelten

$$s[K] = t[K] \quad \Longrightarrow \quad s \simeq t.$$

Zusätzlich muss für jedes Attribute A_i, welches in K vorkommt, gelten: Für jede Instanz R von $\mathscr{S}$ und alle $s \in R$

$$s[A_i] \text{ hat nicht den Wert } \texttt{Null}.$$

Die Kombination aus unique Constraint und not null Constraints garantiert, dass der Primärschlüssel tatsächlich *alle* Elemente einer Relation eindeutig identifizieren kann. Der Fall aus Beispiel 2.7 wird durch die not null Constraints ausgeschlossen.

References Constraints

Fremdschlüssel erlauben uns, Beziehungen zwischen verschiedenen Schemata herzustellen. Damit diese Beziehungen tatsächlich bestehen, benötigen wir bestimmte Integritätsbedingungen auf den Relationen der beteiligten Schemata. Wir werden diese Bedingungen nicht nur für Fremdschlüssel formulieren, sondern sie ein wenig allgemeiner fassen.

Es seien $A_1, \ldots, A_n$ und $B_1, \ldots, B_h$ Attribute. Weiter seien zwei Schemata

$$\mathscr{S}_1 = (A_1, \ldots, A_n) \quad \text{und} \quad \mathscr{S}_2 = (B_1, \ldots, B_h)$$

gegeben. Zusätzlich sei auf $\mathscr{S}_2$ ein unique Constraint $U := (B_{j_1}, \ldots, B_{j_m})$ definiert worden. Ein *references Constraint* von $\mathscr{S}_1$ nach U ist bestimmt durch eine Sequenz

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

von Attributen aus $\mathscr{S}_1$, welche dieselbe Länge hat wie U. Wir sagen dann F *referenziert* U.

Definition 2.10 (Integritätsregel: references Constraint). Gegeben seien ein Relationenschema $\mathscr{S}_2$ mit einem unique Constraint U sowie ein Relationenschema $\mathscr{S}_1$ mit einem references Constraint $F = (A_{i_1}, \ldots, A_{i_m})$ nach U. Für jede Instanz R von $\mathscr{S}_1$ und jede Instanz S von $\mathscr{S}_2$ muss gelten: Für jedes $t \in R$, falls

$$t[A_{i_k}] \text{ ist nicht } \texttt{Null} \text{ für alle } 1 \leq k \leq m,$$

dann gibt es $s \in S$ mit

$$t[F] = s[U].$$

Da jeder primary key Constraint einen unique Constraint impliziert, können wir anstelle von U auch den Primärschlüssel von $\mathscr{S}_2$ verwenden. Da ein Schema nur einen Primärschlüssel besitzen kann, brauchen wir diesen nicht explizit anzugeben. Wir sagen

dann *F referenziert das Schema* $\mathscr{S}_2$ oder eben *F ist ein Fremdschlüssel (foreign key) für* $\mathscr{S}_2$.

Verletzungen der referentiellen Integrität (d. h. Verletzung der Integritätsregel 4) können dann auftreten, wenn neue Tupel in eine Relation eingefügt werden oder bestehende Tupel aus einer Relation gestrichen werden. Im Beispiel 2.5 ist dies der Fall, wenn ein neues Tupel t mit

$$t.[\texttt{FahrerId}] = 4$$

in die Tabelle `Autos` eingefügt wird oder wenn das Tupel

$$(3,\ \texttt{Eva},\ \texttt{Meier})$$

aus der Tabelle `Personen` gelöscht wird.

Jedoch dürfen Attribute in Fremdschlüsseln (oder allgemeiner Attribute, welche in references Constraints vorkommen) den Wert `Null` annehmen, wie das folgende Beispiel zeigt.

Beispiel 2.11. Wir betrachten die Schemata aus Beispiel 2.5, wobei wir einen references Constraint von `FahrerId` nach `PersId` annehmen. Nun fügen wir ein neues Tupel

```
(VW,blau,2015,Null)
```

zu der Relation `Autos` hinzu, so dass wir folgende Tabelle erhalten:

Autos

Marke	**Farbe**	**Baujahr**	**FahrerId**
Opel	silber	2010	1
Opel	schwarz	2010	2
VW	rot	2014	2
Audi	schwarz	2014	3
VW	blau	2015	-

Für den blauen VW ist der Fahrer unbekannt, deshalb hat das entsprechende Attribut den Wert `Null`. Dieser Eintrag verletzt die referentielle Integrität nicht, da die Integritätsregel 4 nur Tupel betrifft, welche in den Attributen des references Constraints nicht den Wert `Null` haben. Damit kann auch in der Gegenwart eines references Constraints ausgedrückt werden, dass keine Referenz vorhanden ist.

Postulat. Das Einhalten der vier Integritätsregeln sollte von einem Datenbanksystem laufend automatisch überprüft werden.

Weiterführende Literatur[3]

1. Codd, E.F.: Derivability, redundancy and consistency of relations stored in large data banks. IBM Research Report, San Jose **RJ599** (1969). Neu erschienen als [3]
2. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM **13**(6), 377–387 (1970). https://doi.org/10.1145/362384.362685
3. Codd, E.F.: Derivability, redundancy and consistency of relations stored in large data banks. SIGMOD Rec. **38**(1), 17–36 (2009). https://doi.org/10.1145/1558334.1558336

[3]Das Relationenmodell geht zurück auf die Arbeiten von Codd [1, 2], welche auch nach über 30 Jahren noch äussert lesenswert sind.