3 Das relationale Datenmodell

3.1 Grundlagen

3.1.1 Einige Definitionen

Kartesisches Produkt:

Seien $W_1, W_2, \ldots, W_n$ beliebige Mengen. Dann ist das kartesische Produkt $W_1 \times W_2 \times \ldots \times W_n$ folgendermaßen definiert:

$$W_1 \times W_2 \times \ldots \times W_n = \left\{ (w_1, w_2, \ldots, w_n) \,\middle|\, w_j \in W_j, j = 1, 2, \ldots, n \right\} .$$

Dabei ist

- $(w_1, w_2, \ldots, w_n)$ ein n-Tupel,
- w_j ist eine (die j-te) Komponente.

Relation (im mathematischen Sinn):

In der Mathematik ist eine Relation als Teilmenge des kartesischen Produkts definiert:

$$X \subseteq W_1 \times W_2 \times \ldots \times W_n .$$

Man sagt

- X ist eine n-stellige Relation über den Mengen $W_1, W_2, \ldots, W_n$ (die auch Wertebereiche oder Domänen genannt werden).
- n wird als Grad der Relation bezeichnet.

Tupel x der Relation X:

Da eine Relation X definitionsgemäß eine Teilmenge des kartesischen Produkts $X \subseteq W_1 \times W_2 \times \ldots \times W_n$ darstellt, ist X eine Menge von Tupeln. Wenn also $x = (x_1, x_2, \ldots, x_n) \in X$, dann muß natürlich $x \in W_1 \times W_2 \times \ldots \times W_n$.

3.1.2 Attribute, Domänen, Tupel und Relationen

Relation (im DB-Sinn):

Die Mengen W_1, W_2, ... des kartesischen Produkts sind nun Attribute a_1, a_2, ...

- Zu einem Attribut a gehört sein Name (also a) und sein Wertebereich (Domäne). Dieser wird durch $\mathrm{dom}(a)$ symbolisiert (vgl.2.1.2).
- Die zu einer Relation (im DB-Sinn) gehörenden Attribute fassen wir zusammen zur Menge $A = \{a_1, a_2, \ldots, a_n\}$. Es wird dabei vorausgesetzt, daß alle Namen *paarweise verschieden* sind*.

* Zwei Domänen $\mathrm{dom}(a_i)$ und $\mathrm{dom}(a_j)$ können aber ohne weiteres identisch sein.

- Obwohl die Attribute einer Relation immer durch ihre Namen identifiziert werden können (vgl. den vorigen Punkt) und ihre Reihenfolge daher einerlei ist, wollen wir im folgenden der Einfachheit halber von einer (zwar beliebigen aber) festen Reihenfolge der Attribute a_1, a_2, ..., a_n in einem Tupel ausgehen[†].
- Der vorige Punkt impliziert natürlich auch eine feste Reihenfolge der Attribute in
$$\mathrm{dom(A)} = \mathrm{dom}(a_1) \times \mathrm{dom}(a_2) \times \ldots \mathrm{dom}(a_n).$$
dom(A) ist die Menge aller Tupel, die man über den Attributen a_1, a_2, ..., a_n bilden kann.

Relationstyp:

Sei $A = \{a_1, a_2, \ldots, a_n\}$ eine Menge von Attributen und Σ eine Menge von (semantischen) Integritätsbedingungen (vgl. 3.3.1). Dann wird durch A und Σ ein Relationstyp festgelegt, symbolisch $(A \,|\, \Sigma)$ bzw. $(a_1, a_2, \ldots, a_n \,|\, \Sigma)$.

Einem *Relationstyp* $(A \,|\, \Sigma)$ kann man auf folgende Weise auch einen Namen R geben:
$$R = (A \,|\, \Sigma) \qquad \text{bzw.} \qquad R = (a_1, a_2, \ldots, a_n \,|\, \Sigma)\,.$$

Zu einem *Relationstyp* gehören somit die *folgenden Bestandteile*:

- R: R ist der Name des Relationstyps
- A: $A = \{a_1, a_2, \ldots, a_n\}$ die Menge der dazugehörigen Attribute, die man auch als das Format von R bezeichent.
- Σ: $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ ist die Menge der zu R gehörenden (semantischen) Integritätsbedingungen.

Analog zu *Datentypen* in höheren Programmiersprachen legt auch ein *Relationstyp* $R = (A \,|\, \Sigma)$ einen Wertebereich fest, der durch val(R) notiert wird. Wenn es keine Integritätsbedingungen gäbe, wäre val(R) die Menge aller Teilmengen X, die man aus dom(A) bilden kann. Wegen der Integritätsbedingungen Σ umfaßt val(R) aber nur diejenigen Teilmengen X von dom(A), die auch die Integritätsbedingungen Σ erfüllen:
$$\mathrm{val}(R) = \{X \subseteq \mathrm{dom}(A) \,|\, \Sigma(X)\}\,.$$

Eine konkrete Relation r vom Typ R (bzw. vom Typ $(A \,|\, \Sigma)$ oder vom Typ $R(A \,|\, \Sigma)$) kann man als *Variable* vom Typ R auffassen. Das wird durch die Schreibweisen

- $r : R$ oder
- $r : (A \,|\, \Sigma)$ oder

[†] Man müßte sonst die ganze Zeit mit dem Codd'schen Konzept der relationship arbeiten, bei dem ein "Tupel" als *Menge von Paaren* definiert ist, wobei jedes dieser Paare aus $(a_i : w_i)$, $w_i \in \mathrm{dom}(a_i)$ besteht. Vgl. dazu [E.F.Codd, "A relational model for large shared data banks", CACM 13, June 1970, 377-387]. Die obige Einschränkung dient also nur zur notationellen Entlastung und zur Vereinfachung der Darstellung.

- $r : R(A \mid \Sigma)$

zum Ausdruck gebracht.

Sei r eine konkrete Relation vom Typ R, also $r : R(A|\Sigma)$. Dann gelten folgende Definitionen:

- $\text{typ}(r) = R$ (oder $(A|\Sigma)$)
- $\text{format}(r) = A$ (das bedeutet, daß $r \subseteq \text{dom}(A)$)

Wenn r eine konkrete Relation vom Typ R ist, also $r : R(A|\Sigma)$, dann ist $r \in \text{val}(R)$. Das heißt, daß

(1) $\text{format}(r) = A$ (also $r \subseteq \text{dom}(A)$)

(2) r erfüllt jede Integritätsbedingung aus Σ. Sei $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$. Dann symbolisieren wir durch $\Sigma(r)$, daß r jede Integritätsbedingung $\sigma_j \in \Sigma$ erfüllt. Analog wird durch die Notation $\sigma_j(r)$ zum Ausdruck gebracht, daß r die Integritätsbedingung σ_j erfüllt.

Wie bereits gesagt, kann $r : (A|\Sigma)$ als Variable (vom Relationstyp R) betrachtet werden. Wenn man sich explizit auf die Ausprägung von r zum Zeitpunkt t bezieht, bringt man das durch die Notation r^t zum Ausdruck (analog zur Entitätsmenge E^t, vgl. 2.1.1). Wenn man hingegen auf die Variableneigenschaft abstellt, spricht man vom Relationsschema r.

Eine Relation wird üblicherweise als (zweidimensionale) *Tabelle* dargestellt. Dabei entsprechen die *Spalten* der Tabelle den *Attributen* der Relation, die *Zeilen* den *Tupeln*, die einzelnen Werte der Tabelle den Attributswerten. Da eine Relation als Menge von Tupeln definiert ist,

- kann in der Tabelle *keine Zeile mehrfach* vorkommen,
- ist die *Reihenfolge* der Zeilen in der Tabelle *ohne Bedeutung*
- ist die *Reihenfolge* der Spalten in der Tabelle *ohne Bedeutung*, weil die Spalten durch die Namen der Attribute eindeutig identifiziert werden können.

3.1.3 Schlüssel einer Relation

Im ER-Modell wurde ein Schlüssel für einen Entitätstyp als Attributskombination definiert, die *identifizierend* und *minimal* ist (vgl. 2.1.3). Das gilt auch für Relationen:

Eigenschaften eines Schlüssels:

Sei $r : R(A|\Sigma)$ ein Relationsschema vom Typ R und sei $K \subseteq A$ eine Menge von Attributen. K ist Schlüssel des Relationstyps R bzw. des Relationsschemas r, wenn K

(1) *identifizierend*

(2) *minimal*

ist. Unter Verwendung der *Projektion* (eines Tupels x bzw. einer Relation r auf eine Teilmenge seiner bzw. ihrer Attribute, vgl. 3.2.2) kann man das folgendermaßen formal definieren:

(1) K ist identifizierend, wenn für jede mögliche Ausprägung $X \in \text{val}(R)$ von r und für alle Tupel $x, y \in X$ gilt:

$$(x.K = y.K) \implies (x = y)\,.$$

(2) K ist minimal, wenn es für jede Teilmenge $K' \subset K$ eine Ausprägung $X \in \text{val}(R)$ von r mit zwei verschiedenen Tupeln $x, y \in X$ $(x \neq y)$ gibt, so daß:

$$x.K' = y.K'$$

Das bedeutet, daß man kein Attribut von K weglassen kann, ohne daß K seine Identifikationseigenschaft verliert.

Schlüsselattribut — Nichtschlüsselattribut (NSA):

Ein Attribut $a \in A$ ist entweder Schlüsselattribut oder Nichtschlüsselattribut (NSA). a ist Schlüsselattribut, wenn es einen Schlüssel K gibt, der a enthält (also $a \in K$)[‡]; Sonst (also wenn kein Schlüssel a enthält) ist a Nichtschlüsselattribut (NSA).

Primärschlüssel (PS):

Das relationale Modell verlangt, daß ein Schlüssel als Primärschlüssel (PS) ausgezeichnet wird. Der Primärschlüssel ist gewissermaßen der offizielle Arbeitsschlüssel eines Relationsschemas*. Im relationalen Modell ist die Festlegung des PS eine spezielle Integritätsbedingung (die zu Σ gehört). Der Primärschlüssel stellt gewissermaßen die logische Tupeladresse dar. Der PS wird häufig durch *Unterstreichen* der dazugehörigen Attribute gekennzeichnet.

3.1.4 NULL-Werte

In der Praxix kann es vorkommen, daß ein Wert für ein Attribut (noch) *nicht bekannt* oder *nicht anwendbar* ist. Für solche Fälle ist der sogenannte NULL-Wert (null value) vorgesehen. NULL ist dabei eine spezielle Markierung für fehlende (oder nicht anwendbare etc.) Werte, die sich von allen "richtigen" Werten der Domäne unterscheidet (insbesondere ist NULL verschieden von der Zahl 0 oder vom Leerzeichen).

- NULL-Werte sind eine für die Praxis notwendige Erweiterung des relationalen Modells.
- Attribute, die zum Primärschlüssel gehören, dürfen niemals einen NULL-Wert annehmen (die *logische Tupeladresse* muß immer *wohldefiniert* sein).

[‡] Achtung: Es kann mehrere Schlüssel(kandidaten) geben!

* Wie bereits erwähnt, gibt es in der Regel mehrere Schlüssel, die auch als *Schlüsselkandidaten* bezeichnet werden. Man muß sich dann für einen davon als *Primärschlüssel* entscheiden. Die übrigen werden *Alternativschlüssel* genannt.

- `NULL`-Werte bereiten Probleme bei der Auswertung von Ausdrücken, vor allem bei *Boolschen Ausdrücken* (für die eine *dreiwertige Logik* erforderlich wird).

3.1.5 Die erste Normalform (1 NF)

Sei $r : (A \mid \Sigma)$. Man sagt: r ist in 1. Normalform (1 NF), wenn jedes Attribut $a \in A$ atomar ist. Ein Attribut ist dann atomar, wenn alle Elemente $\alpha \in \mathrm{dom}(a)$ der Domäne des Attributs einzelne ("atomare", "skalare", "nicht weiter strukturierte") Werte darstellen.

Durch die 1 NF werden insbesondere *mengenwertige* Attribute ausgeschlossen[†].

3.1.6 Modellierung der Realwelt im rel. Datenmodell

Ein Beispiel, indem auch schon vorbereitet wird, daß $1 : n$ Beziehungstypen des ER-Modells einfacher ins relationale Modell übertragen werden können, als $n : m$ Beziehungstypen (vgl. 3.4.3) Auch die Fremdschlüssel (vgl. 3.1.7) werden schon vorbereitet.

3.1.7 Fremdschlüssel

Sei $r_1 : (A \mid \Sigma_1)$ und $r_2 : (B \mid \Sigma_2)$. Sei $FS \subseteq A$ eine Menge von Attributen von r_1 und PS der Primärschlüssel des Relationsschemas r_2. Mit Hilfe des Fremdschlüssels FS kann man eine $(n : 1)$ Beziehung zwischen r_1 und r_2 herstellen. r_1 wird dabei *referenzierende* Relation und r_2 *referenzierte* Relation genannt. Voraussetzungen dazu sind:

- $\mathrm{dom}(FS) = \mathrm{dom}(PS)$. Das heißt, die Wertebereiche von FS und PS müssen übereinstimmen.
- $r_1.FS \subseteq r_2.PS$. Das heißt, jeder in r_1 auftretende Fremdschlüsselwert kommt auch als Primärschlüsselwert in r_2 vor. Damit referenziert jedes Tupel von r_1 ein Tupel aus r_2.

In Analogie zum Primärschlüssel als logischer Tupeladresse (von r_2) kann man somit einen Fremdschlüssel als logischen Pointer auffassen, mit Hilfe dessen jedes Tupel von r_1 auf ein Tupel von r_2 zeigt. So wie die Festlegung eines Primärschlüssels im relationalen Modell eine Integritätsbedingung darstellt, ist es auch die Festlegung eines Fremdschlüssels. Allerdings sind jetzt zwei Tabellen involviert (die referenzierende und die referenzierte) und die entsprechende Integritätsbedingung ist daher eine tabellenübergreifende Integritätsbedingung.

Bemerkungen:

[†] In 3.1.5 (3|5) – (5|5) werden drei Möglichkeiten diskutiert, wie man Relationen, die wegen mengenwertiger Attribute nicht in 1 NF sind, in 1 NF bringen kann, nämlich: 'Attribut-Verdopplung', 'Tupel-Verdopplung', 'Zerlegung in 2 Relationen'. Im allgemeinen nur 3. Variante empfehlenswert.

- Ebenso wie ein "gewöhnlicher" Pointer kann auch ein Fremdschlüssel als logischer Pointer nirgendwohin zeigen (der Pointer wird dann zum `NULL`-Pointer). Im relationalen Modell wird das durch den `NULL`-Wert zum Ausdruck gebracht[‡].
- Wie die obigen Bedingungen zeigen ist Namensgleichheit der Attribute in FS und PS nicht erforderlich, aber erlaubt und im allgemeinen sogar hilfreich.
- Es ist zulässig, daß ein Relationsschema $r_1 : R_1$ mehrere Fremdschlüssel auf die Relationsschemata $r_2 : R_2$, ..., $r_n : R_n$ aufweist. Es können sich dabei auch mehrere Fremdschlüssel auf das gleiche Relationsschema beziehen.
- Es ist auch zulässig, daß das referenzierte Relationsschema mit dem referenzierenden Relationsschema identisch ist ("Selbstbezug").

[‡] Selbstverständlich ist das nur dann zulässig, wenn es nicht durch eine Integritätsbedingung ausgeschlossen wird, beispielsweise weil ein zu FS gehörendes Attribut auch zum Primärschlüssel PS der referenzierenden Tabelle gehört.

3.2 Relationale Algebra

Die Algebra beschäftigt sich mit algebraischen Strukturen (eben Algebren) wie Gruppen, Ringen, Körpern, Verbänden. Eine solche Algebra besteht dabei im wesentlichen aus einer nichtleeren Menge (z.B. der Menge der natürlichen Zahlen) und Operationen auf dieser Menge (z.B. Addition). Im Falle der Relationenalgebra handelt es sich bei der Menge um eine Menge von Relationen und bei den Operationen um die auf Relationen anwendbaren Operationen.

Der Zweck der relationalen Algebra besteht darin, die auf Relationen anwendbaren Operationen zu definieren. Diese Operationen sind in erster Linie dazu da, um Abfragen (Queries) formulieren zu können. Somit kann man auch sagen, daß durch die relationale Algebra die Abfragemöglichkeiten im relationalen Modell — und damit die eines relationalen DBMS — definiert werden.

Die Operationen der relationalen Algebra können nach zwei Gesichtspunkten eingeteilt werden:

- Nach der Art der Operation
  - Mengenoperationen
  - spezifische relationale Operationen
- Nach der Anzahl der Operanden
  - binäre Operationen
  - unäre Operationen

Im folgenden wird von der Einteilung nach der Art der Operationen ausgegangen.

3.2.1 Mengenoperationen

Sei $r_1 : (A \mid \Sigma_1)$ und $r_2 : (B \mid \Sigma_2)$. Da eine Relation eine *Menge von Tupeln* (aus $\mathrm{dom}(A)$ bzw. $\mathrm{dom}(B)$) darstellt, kann man natürlich die klassischen Mengenoperationen *Vereinigung*, *Durchschnitt* und *Differenz* anwenden. Das geht allerdings nur unter der Voraussetzung, daß r_1 und r_2 vereinigungskompatibel sind:

r_1 und r_2 sind genau dann vereinigungskompatibel, wenn $\mathrm{format}(r_1) = \mathrm{format}(r_2)$, oder (was auf dasselbe hinausläuft), wenn $A = B$. Unter dieser Voraussetzung können wir definieren:

Vereinigung: $r_1 \cup r_2 = \left\{ x \,\middle|\, x \in r_1 \vee x \in r_2 \right\}$

Durchschnitt: $r_1 \cap r_2 = \left\{ x \,\middle|\, x \in r_1 \wedge x \in r_2 \right\}$

Differenz: $r_1 \setminus r_2 = \left\{ x \,\middle|\, x \in r_1 \wedge x \notin r_2 \right\}$

Das kartesische Produkt besteht aus allen Tupeln, die durch Verkettung der Tupel von $r_1 : (A = \{a_1, a_2, \ldots, a_n\} \mid \Sigma_1)$ mit denen von $r_2 : (B = \{b_1, b_2, \ldots, b_m\} \mid \Sigma_2)$ gebildet werden können[†]:

(kartesisches) Produkt: $r_1 \times r_2 = \big\{x = x_1x_2 \,\big|\, x_1 \in r_1 \wedge x_2 \in r_2\big\}$.

Dabei übernimmt ("erbt") die Ergebnisrelation die Attribute von A und B. Wenn A und B keine Attribute mit gleichen Namen haben, ist das völlig problemlos, weil dann alle Attribute in $\{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m\}$ eindeutige Namen haben. Wenn es andererseits Attribute mit gleichen Namen gibt, kann man die Attributsnamen durch Verwendung qualifizierter Attributsnamen

$$\{r_1.a_1,\, r_1.a_2,\, \ldots,\, r_1.a_n, \quad r_2.b_1,\, r_2.b_2,\, \ldots,\, r_2.b_m\}$$

eindeutig machen. Dabei wird der Attributsname a mit dem Namen des entsprechenden Relationsschemas r qualifiziert $(r.a)$.*

Bemerkung:

Die Ergebnisrelationen aller Mengenoperationen erben auf natürliche Weise die Attribute von ihren Operandenrelationen. Bezüglich der Domänen gibt es dabei keinerlei Probleme. Was die Attributsnamen anlangt, muß man für das kartesische Produkt die Eindeutigkeit gegebenenfalls durch Verwendung qualifizierter Namen herstellen.

3.2.2 Relationenspezifische Operationen

Die spezifischen relationalen Operationen sind *Projektion*, *Selektion* und *Verbund (Join)*.

Projektion:

Mit Hilfe der Projektion kann man Attribute aus einer Relation auswählen. Tabellenmäßig läuft das auf eine *Auswahl von Spalten* und somit auf die Bildung einer *vertikalen Teiltabelle* hinaus. Die *Projektion* ist eine *unäre* Operation: Es gibt also keine Komatibilitätsprobleme.

Sei $r : (A \,|\, \Sigma)$ und sei $L \subseteq A$ eine Teilmenge der Attribute. Sei $x \in r$ ein Tupel von r. Dann definieren wir zunächst die

Projektion von x auf die Attribute in L (vgl. 3.1.3): Diese wird durch

$$x.L$$

[†] Es gilt natürlich: Der *Grad des Produkts* ist die Summe der Grade der Faktoren. Die *Kardinalität des Produkts* ist das Produkt der Kardinalitäten.

* Manchmal wird als Kompatibilitätsbedingung verlangt, daß es keine gleichen Attributsnamen in A und B geben darf. Diese Einschränkung ist ziemlich restriktiv und kann durch Verwendung der qualifizierten Namen vermieden werden.

symbolisiert. Es ergibt sich dabei das Teiltupel von x, welches nur die Werte der Attribute in L umfaßt. Damit können wir definieren die

Projektion von r auf die Attribute in L: Diese wird durch

$$\pi_{[L]}r \quad \text{oder (eleganter) durch} \quad r.L$$

symbolisiert. Das Ergebnis

$$\pi_{[L]}r = r.L = \big\{x.L \,\big|\, x \in r\big\}$$

besteht aus der Menge der auf L projizierten Tupel von r.

Selektion:

Mit Hilfe der Selektion kann man Tupel aus einer Relation auswählen, nämlich diejenigen Tupel, die eine Bedingung b — die Selektionsbedingung — erfüllen. Tabellenmäßig läuft das auf eine *Auswahl von Zeilen* und somit auf die Bildung einer *horizontalen Teiltabelle* hinaus. Die *Selektion* ist eine *unäre* Operation: Es gibt also keine Komatibilitätsprobleme[‡].

Sei $r : (A \,|\, \Sigma)$ und sei b eine Selektionsbedingung (ein logischer (boolscher) Ausdruck), in dem neben Konstanten auch Attribute a_1, a_2, …, $a_n \in A$ vorkommen dürfen. Dann definieren wir die

Selektion von r (gemäß der Selektionsbedingung b):

$$\sigma_{[b]}r = \big\{x \in r \,\big|\, x \text{ erfüllt } b\big\}$$

Das Ergebnis besteht also aus allen Tupeln, welche die Bedingung b erfüllen.

Verbund (Join):

Mit Hilfe des Verbundes (Join) kann man zwei Relationen zu einer neuen Relation verknüpfen, wobei die folgenden zwei Varianten unterschieden werden:

- natürlicher Verbund (vgl. auch 3.5)
- Verbund mit einer Verbundbedingung

Natürlicher Verbund:

Seien $r_1 : (A \,|\, \Sigma_1)$ und $r_2 : (B \,|\, \Sigma_2)$. Dann ist der *natürliche Verbund* $r_1 * r_2$ definiert als

$$r_1 * r_2 = \big\{x \in \mathrm{dom}(A \cup B) \,\big|\, x.A \in r_1, x.B \in r_2\big\} \ .$$

[‡] In SQL wird (fast) die volle Funktionalität der relationalen Algebra (also viel mehr als nur die Selektion) durch die `SELECT`-Abfrage abgedeckt. Daher wurde verschiedentlich (z.B. von *C.J.Date*) angeregt, statt 'Selektion' die Bezeichnung 'Restriktion' zu verwenden.

Die Ergebnisrelation umfaßt die *privaten Attribute* von r_1 (also $A \setminus B$), die *gemeinsamen Attribute* von r_1 und r_2 (also $A \cap B$) sowie die *privaten Attribute* von r_2 (also $B \setminus A$). Unter den jeweils privaten Attributen darf es keine gemeinsamen Attributsnamen geben. Es werden dabei alle in den gemeinsamen Attributen übereinstimmenden Tupel von r_1 und r_2 *verschmolzen* (das bedeutet, daß die Werte der — sowieso übereinstimmenden — gemeinsamen Attribute im Resultat nur einmal berücksichtigt werden).

Bemerkungen:

- Wenn es keine gemeinsamen Attribute gibt, fällt der natürliche Verbund mit dem (kartesischen) Produkt zusammen:

$$(A \cap B = \emptyset) \implies (r_1 * r_2 = r_1 \times r_2)\,.$$

- Wenn es nur gemeinsame Attribute gibt, fällt der natürliche Verbund mit dem Durchschnitt zusammen:

$$(A = B) \implies (r_1 * r_2 = r_1 \cap r_2)\,.$$

- Der natürliche Verbund ist *assoziativ*:

$$(r_1 * r_2) * r_3 = r_1 * (r_2 * r_3)\,.$$

- Der natürliche Verbund ist *kommutativ* (bis auf die Reihenfolge der Attribute, die sowieso ohne Bedeutung ist, vgl. 3.1.2):

$$r_1 * r_2 = r_2 * r_1\,.$$

Verbund mit einer Verbundbedingung:
Beim Verbund mit Verbundbedingung gehen wir vom (kartesischen) Produkt

$$r_1 \times r_2$$

aus, das bereits in 3.2.1 besprochen worden ist[†]. Damit können wir den *Verbund mit Verbundbedingung* definieren als

$$r_1 *_{[b]} r_2 = \sigma_{[b]}\,(r_1 \times r_2)\ .$$

[†] Dabei muß man in der Ergebnisrelation erforderlichenfalls die Attribute durch Verwendung der qualifizierten Attributsnamen

$$r_1.a_1,\, r_1.a_2,\, \ldots,\, r_1.a_n, \quad r_2.b_1,\, r_2.b_2,\, \ldots,\, r_2.b_m$$

eindeutig machen, wobei der Attributsname a mit dem Namen des entsprechenden Relationsschemas r qualifiziert ($r.a$) wird (vgl. 3.2.1).

Dabei ist die *Verbundbedingung* b ein logischer (boolscher) Ausdruck in dem (erforderlichenfalls qualifizierte) Attributsnamen von $r1$ und $r2$ verwendet werden können. Die Verbundbedingung wird als Selektionsbedingung auf das erweiterte (kartesische) Produkt angewendet.

Der sogenannte θ-Verbund ist ein Spezialfall des Verbundes mit Verbundbedingung. Die Verbundbedingung ist hier ein *einfacher Vergleich* $a\,\theta\,b$ zwischen (vergleichbaren) Attributswerten aus je einer der beiden Relationen, $a \in A$, $b \in B$ und $\theta \in \{=, \neq, <, \leq, >, \geq\}$. So spricht man beispielsweise vom *Equi-Join*, wenn θ der $=$-Vergleichsoperator ist.

Weitere Join-Varianten:

- Equi-Join (s.o.): Spezialfall des Verbunds mit einer Verbundbedingung, wobei die Verbundbedingung die Form $a_1 = b_1 \wedge a_2 = b_2 \wedge \ldots \wedge a_n = b_n$ hat.
- Semi-Join: Natürlicher Verbund mit anschließender Projektion auf Attribute einer der beiden Relationen: $(r_1 * r_2).A$
- Left-Outer-Join: Ergebnis des natürlichen Verbundes wird ergänzt durch Tupel der linken Relation, die nicht zum Zug gekommen sind (für die Attribute $B \setminus A$ werden dabei `NULL`-Werte eingetragen).
- Right-Outer-Join: Ergebnis des natürlichen Verbundes wird ergänzt durch Tupel der rechten Relation, die nicht zum Zug gekommen sind (für die Attribute $A \setminus B$ werden dabei `NULL`-Werte eingetragen).
- Full-Outer-Join: Kombination von Left- und Right-Outer-Join.
- Union-Join: Hat die gleichen Attribute wie der natürliche Verbund, also $A \cup B$. Tupel von r_1 werden um Attribute $B \setminus A$ erweitert. Tupel von r_2 werden um Attribute $A \setminus B$ erweitert. Für die ergänzten Attribute werden jeweils `NULL`-Werte eingetragen. Anschließend werden die erweiterten Tupel von r_1 und r_2 vereinigt.

3.2.3 Ausdrücke der Relationenalgebra

In den Operationen der Relationenalgebra (vgl. 3.2.2) können als Operanden nicht nur Basisrelationen vorkommen (das sind Relationen, die durch *Relationsschemata* definiert sind), sondern auch abgeleitete Relationen. Das sind Relationen, die durch einen Ausdruck der Relationenalgebra definiert sind. Schließlich gelangt man auf diese Weise zu beliebig tief verschachtelten Ausdrücken der relationalen Algebra. In 3.2.3 wird eine kontextfreie Grammatik für die Regeln solcher Ausdrücke (beispielhaft) angegeben.

3.3 Datenbeschreibung und Datenmanipulation

Bei jedem logischen (abstrakten) Datenmodell (z.B. dem relationalen) muß man ebenso wie bei einer konkreten Datenbanksprache (z.B. SQL) die Aspekte der Datenbeschreibung und der Datenmanipulation unterscheiden. Die entsprechenden Teilsprachen sind:

- DDL (data definition (oder description) language): Konstrukte zur *Definition/ Beschreibung* der Daten und ihrer Strukturen.
- DML (data manipulation language): Konstrukte zur *Manipulation* der Daten. Dabei müssen zwei Arten von "Manipulationen" unterschieden werden:
  - Abfragen: Operationen zur Formulierung von Abfragen (Queries). Dadurch kommt es zu keiner Änderung des Status der Datenbank.
  - Mutationen: Operationen zur Eingabe, Löschung, und Änderung von Daten. Dadurch ändert sich der Status der Datenbank.

Unter dem relationalen Modell handelt es sich bei den *Daten* und ihren *Strukturen* natürlich um Relationen (im DB-Sinn). Zur Definition der DML* gibt es zwei Ansätze, nämlich:

- Eine auf Relationen basierende Algebra (prozedural)
  - Relationenalgebra (vgl. 3.2)
- Zwei auf der Prädikatenlogik erster Stufe basierende Kalküle (deskriptiv)
  - Relationenkalkül/Tupelkalkül
  - Domänenkalkül

Nicht Satz- / Tupel- / Record-weise Verarbeitung, sondern mengenorientierte Operationen (Menge von Tupel), kein Navigieren, Anwendung einer Operation auf Relation(en) liefert wieder eine Relation!

Die Operationen des relationalen Modells sind mengenorientiert: Die Anwendung einer Operation auf Relationen(en) ergibt eine Relation — also eine Menge von Tupeln. Demgegenüber waren die früheren logischen Datenmodelle nur Satz- (Record-, Tupel-) orientiert: Man mußte durch die DB navigieren um alle Sätze zu sammeln, die zum Ergebnis der Abfrage gehören.

In diesem Vorspann zu 3.3 wurden im wesentlichen die Informationen aus 3.3.3 zusammengefaßt. In den folgenden Abschnitten '3.3.1 Schema' und '3.3.2 Sichten' geht es darum, wie die Abbildung des konzeptuellen (logischen) bzw. der externen Schemata (Benutzersichten) des 3-Ebenen-Architektur-Konzepts (vgl. 1.6.2) auf die Strukturen des relationalen Modells bewerkstelligt wird.

* Strengenommen geht es bei diesen Ansätzen nur um die Definition der Abfragemöglichkeiten. Zur formalen Definition der Mutationen müßte man noch eine *relationale Wertzuweisung* dazunehmen.

3.3.1 Schema

Das konzeptuelle Schema (also die Beschreibung der gesamten Daten und ihrer Zusammenhänge, typischerweise im E(E)R-Modell, vgl. 1.6.2) muß in eine Menge von *Basisrelationen*† (inklusive jeweils dazugehörender Integritätsbedingungen) und eine Menge von *relationenübergreifenden Integritätsbedingungen* übersetzt werden.

Die **logische Ebene** einer **relationalen Datenbank** besteht somit:

- Typ-Ebene:
  - aus einer Menge von Relationsschemata $\left\{ r_i \colon (A_i \mid \Sigma_i) \;\middle|\; i = 1, \ldots, n \right\}$
  - aus einer Menge Σ_{DB} von relationenübergreifenden Integritätsbedingungen
- Ausprägungsebene:
  - Zu jedem Relationsschema existiert eine (1 NF) Relation vom entsprechenden Typ.
  - Alle Relationen in ihrer Gesamtheit erfüllen (neben den Integritätsbedingungen in Σ_i, $i = 1, \ldots, n$) alle Integritätsbedingungen in Σ_{DB}.

Bemerkungen:

- Alle Relationen gleichberechtigt
- Alle Operationen auf jede Relation anwendbar (kein *Navigieren*, keine *Einstiegspunkte*)
- Auch ein Relationsschema kann geändert / gelöscht werden (aber typischerweise nicht periodisch und eher selten)
- Zusätzliche Angaben über *Zugriffsberechtigungen* und zur *Integritätssicherung* notwendig

3.3.2 Sichten

Für die externen Schemata (ein externes Schema ist ein Ausschnitt aus dem konzeptuellen Schema für eine spezielle Anwendung, auch *Benutzersicht* oder *View* genannt, vgl. 1.6.2) kann man *Sichten (Views) über Basisrelationen* definieren:

Die **Benutzerebene** einer **relationalen Datenbank** besteht somit aus einer Menge von *Sichten* oder *Views*, die im Prinzip auf folgende Weise definiert werden können:

- *Sichtname: Operationsteil*

Häufig werden synonym auch die Bezeichnungen

- *Viewname: Viewformel*

verwendet. Dabei ist der *Operationsteil* (die *Viewformel*) eine Vorschrift zur Ableitung einer *abgeleiteten* Relation (Sicht, View, auch: *virtuelle Relation*). Diese

† Die Basisrelationen entsprechen den in der DB tatsächlich gespeicherten Relationen.

Vorschrift hat die Form eines relationalen Ausdrucks und die abgeleitete Relation ist einfach als Ergebisrelation dieses Ausdrucks definiert. Diese virtuellen Relationen stellen somit das Gegenstück zu den Basisrelationen dar.

Verwendung von Sichten (Views):

- Sichten (Views) können wie Basisrelationen in Abfragen verwendet werden.
- Bezüglich der Mutationen hängt es von der Viewformel ab, ob der View mutierbar ist oder nicht. Mutierbare Views sind wie Basisrelationen mutierbar. Die Mutationen wirken sich in diesem Fall auf die letztlich zugrundeliegenden Basisrelationen aus.

3.4 Übertragung des ER-Modells in das relationale Datenmodell

Ein Datenschema liegt in Form eines ER-Diagramms vor und soll in eine Menge von Relationsschemata des relationalen Modells übertragen werden. Das Problem besteht darin, daß es

- im ER-Modell zwei Konstrukte gibt (nämlich Entitäten und Beziehungen),

während es

- im relationalen Modell nur ein Konstrukt gibt (nämlich Relationen).

In diesem Abschnitt wird gezeigt, wie man Entitäts- (inklusive schwacher Entitäten und Generalisierungen) und Beziehungstypen auf Relationsschemata (inklusive Fremdschlüsseln) übertragen kann.

3.4.1 Übertragung von Entitätstypen

"gewöhnliche" Entitätstypen[‡]:

Sei $E : <A>$ ein Entitätstyp mit dem Primärschlüssel $P \subseteq A$.

Dann wird E in ein Relationsschema $r_E : R_E$ übertragen, wobei

$$R_E = \left(A \,\middle|\, \{P \text{ ist Primärschlüssel}\}\right).$$

schwache Entitätstypen (vgl. 2.1.8):

Sei $W : <A>$ ein schwacher Entitätstyp mit dem (partiellen) Schlüssel $S \subseteq A$.

Die dazugehörigen übergeordneten (starken) Entitäten (von denen Identifikationsabhängigkeit besteht) sind:

$$E_1 : <A_1>,\ E_2 : <A_2>,\ \ldots,\ E_k : <A_k>$$

jeweils mit den Primärschlüsseln:

$$P_1 \subseteq A_1,\ P_2 \subseteq A_2,\ \ldots,\ P_k \subseteq A_k.$$

Dann wird W in ein Relationsschema $r_W : R_W$ übertragen, wobei

$$R_W = \left(A\, P_1 P_2 \ldots P_k \,\middle|\, \Sigma_W\right),$$

$$\Sigma_W = \{S\, P_1 P_2 \ldots P_k \text{ ist Primärschlüssel}\}$$

[‡] Darunter wollen wir Entitätstypen verstehen, die weder *schwache Entitätstypen* noch *Unterentitätstypen* sind.

und $r_W.P_1$, ..., $r_W.P_k$ sind Fremdschlüssel auf die Relationsschemata r_1, ..., r_k, die den starken Entitätstypen entsprechen:

$$\begin{aligned} r_W.P_1 &\subseteq r_1.P_1, \\ r_W.P_2 &\subseteq r_2.P_2, \\ &\ldots\ldots \\ r_W.P_k &\subseteq r_k.P_k. \end{aligned}$$

Oben und im weiteren wird die Kurznotation AB für $A \cup B$ verwendet, analog aB für $\{a\} \cup A$, wobei A, B Attributsmengen sind und a ein Attribut ist.

3.4.2 Übertragung von Generalisierungen*

Oberentitätstyp: $O : <A>$ mit Primärschlüssel $P \subseteq A$

Unterentitätstypen: $U_1 : <A_1>, U_2 : <A_2>, \ldots, U_k : <A_k>$†

Zur Übertragung der Generalisierung / Spezialisierung gibt es die folgenden vier Ansätze:

1) Der Oberentitätstyp O wird ganz gewöhnlich übernommen, also Übertragung von O in ein *Relationsschema* $r_O : R_O$, wobei

$$R_O = \left(A \,\middle|\, \{P \text{ ist Primärschlüssel}\}\right).$$

Für jeden Unterentitätstyp U_i wird ein *Relationsschema* $r_i : R_i$ mit

$$R_i = \left(PA_i \,\middle|\, \{P \text{ ist Primärschlüssel}\}\right)$$

eingerichtet. Dabei ist der *Primärschlüssel* P gleichzeitig *Fremdschlüssel* auf die Relation R_O des Oberentitätstyps: $r_i.P \subseteq r_O.P$.

Bemerkung: Dieser (Standard-) Ansatz kann immer verwendet werden und sollte deshalb im allgemeinen gewählt werden.

2) Für den Oberentitätstyp O wird *kein eigenes Schema* erstellt.

Für jeden Unterentitätstyp U_i wird ein *Relationsschema* $r_i : R_i$ mit

$$R_i = \left(AA_i \,\middle|\, \{P \text{ ist Primärschlüssel}\}\right)$$

---

* Wir beschränken uns hier auf die Darstellung von *Einfachgeneralisierungen*, also keine weitere Spezialisierung von Untertypen.

† Dabei sind A_1, A_2, ..., A_k die speziellen Attribute der Untertypen, die Attribute A des Obertyps (inklusive dem Primärschlüssel P) werden vom Obertyp geerbt (vgl. 2.2.2).

erstellt.

Bemerkung: Dieser Ansatz scheidet allerdings aus, wenn die *Spezialisierung* nur *partiell* ist (vgl. 2.2.2). Bei *überlappender Spezialisierung* käme es zur mehrfachen Speicherung der generellen Attribute.

3) Für alle Entitätstypen O, U_1, U_2, ..., U_k zusammen wird ein *einziges Relationsschema* $r : R$ mit

$$R = \big(A\, A_1\, A_2 \ldots A_k\, t \,\big|\, \{P \text{ ist Primärschlüssel}\}\big)$$

erstellt, wobei t ein zusätzliches (Selektor-) Attribut zur Angabe des jeweiligen Untertyps ist.

Bemerkung: Bei diesem Ansatz sind `NULL`-Werte für die jeweils nicht zutreffenden Spezialattribute notwendig. Der Ansatz kann überdies nur verwendet werden, wenn die *Spezialisierung disjunkt* ist (vgl. 2.2.2).

4) Für alle Entitätstypen O, U_1, U_2, ..., U_k zusammen wird ein *einziges Relationsschema* $r : R$ mit

$$R = \big(A\, A_1\, A_2 \ldots A_k\, \{t_1, t_2, \ldots, t_k\} \,\big|\, \{P \text{ ist Primärschlüssel}\}\big)$$

erstellt, wobei $t_1, t_2, \ldots, t_k$ zusätzliche 0-1-Attribute sind. t_j gibt an, ob die Entität zum Untertyp j gehört oder nicht.

Bemerkung: Auch bei diesem Ansatz sind `NULL`-Werte für die jeweils nicht zutreffenden Spezialattribute notwendig. Im Gegensatz zu Ansatz 3) kann dieser Ansatz aber auch für *nicht disjunkte Spezialisierungen* (vgl. 2.2.2) eingesetzt werden.

3.4.3 Übertragung von Beziehungstypen

Im folgenden wird zunächst die Übertragung binärer Beziehungstypen ($(1:1)$, $(1:n)$, $(n:m)$) besprochen. Anschließend wird auf die Übertragung von Beziehungstypen höheren Grades eingegangen.

$(1:1)$-Beziehungen:

- Ein $(1:1)$-Beziehungstyp wird im allgemeinen nicht zu einer eigenen Relation.
- Vielmehr wird die Information (Fremdschlüssel auf den anderen Entitätstyp, etwaige Beziehungsattribute) bei einer der beiden den Entitätstypen entsprechenden Relationen "angehängt".

$(1:n)$-Beziehungen:

- Auch ein $(1 : n)$-Beziehungstyp wird im allgemeinen nicht zu einer eigenen Relation.

- Die Information (Fremdschlüssel auf den anderen Entitätstyp, etwaige Beziehungsattribute) wird an die Relationen "angehängt", die dem Entitätstyp an der mit n beschrifteten Kante der Beziehung entspricht[‡].

$(n:m)$-Beziehungen:

- Ein $(n:m)$-Beziehungstyp muß in ein eigenes Relationsschema B übertragen werden.
- Diese enthält die Schlüssel K_1, K_2 der beteiligten Entitätstypen und etwaige Beziehungsattribute. $K_1\,K_2$ wird zum Primärschlüssel von B. Außerdem sind K_1 und K_2 Fremdschlüssel, welche die den beiden Entitätstypen entsprechenden Relationen referenzieren.

Bemerkungen:

- Die Vorgangsweise für $(n:m)$-Beziehungstypen läßt sich natürlich auch für $(1:1)$- und $(1:n)$-Beziehungstypen anwenden. In diesen Fällen kann man sich aber die zusätzliche "Beziehungsrelation" B ersparen.
- Der $(1:1)$-Fall ist ein Spezialfall des $(1:n)$-Falles: Man kann sich jetzt gewissermaßen aussuchen, "welchen der beiden 1-er man als n auffassen will".
- Die dargestellte Vorgangsweise für *"gewöhnliche" binäre Beziehungen* kann ebenso bei *binären* Selbstbezügen ("rekursiven" Beziehungen) angewendet werden. Es wird dabei empfohlen, die entsprechenden *Rollennamen* (vgl. 2.1.6) in die Namen hinzugefügter Attribute aufzunehmen.

Beziehungen höheren Grades (als 2):

Sei $B : < E_1, E_2, \ldots, E_k \,/\, A_Z >$ ein Beziehungstyp vom Grad $k > 2$ (vgl. 2.1.4 und 2.1.7) und seien

$$\begin{aligned}
r_1 : R_1 \quad &\text{mit} \quad R_1 = \left(A_1 \,\middle|\, \Sigma_1\right), \quad (P_1 \text{ ist Primärschlüssel}) \in \Sigma_1, \\
r_2 : R_2 \quad &\text{mit} \quad R_2 = \left(A_2 \,\middle|\, \Sigma_2\right), \quad (P_2 \text{ ist Primärschlüssel}) \in \Sigma_2, \\
&\vdots \\
r_k : R_k \quad &\text{mit} \quad R_k = \left(A_k \,\middle|\, \Sigma_k\right), \quad (P_k \text{ ist Primärschlüssel}) \in \Sigma_k
\end{aligned}$$

die den beteiligten Entitäten E_1, E_2, …, E_k entsprechenden Relationsschemata.

Dann wird B in ein eigenes Relationsschema $r_B : R_B$ übertragen, wobei

$$R_B = \left(P_1\, P_2 \ldots P_k\, A_z \,\middle|\, \Sigma_B\right).$$

[‡] In diese Richtung kann es höchstens eine Beziehung geben, so daß man die Beziehung durch einen Fremdschlüssel adäquat darstellen kann.

$r_B.P_1$, ..., $r_B.P_k$ sind Fremdschlüssel auf die Relationsschemata r_1, ..., r_k, die den beteiligten Entitätstypen entsprechen:

$$\begin{aligned} r_B.P_1 &\subseteq r_1.P_1, \\ r_B.P_2 &\subseteq r_2.P_2, \\ &\ldots\ldots \\ r_B.P_k &\subseteq r_k.P_k. \end{aligned}$$

Wenn der Entitätstyp E_j bezüglich B eine Mindestkardinalität von 1 hat ("zwingende Mitgliedschaft" von E_j an der Beziehung, vgl.2.2.1), muß zusätzlich die relationenübergreifende Integritätsbedingung ("Fremdschlüssel auch in die andere Richtung")

$$r_j.P_j \subseteq r_B.P_j$$

formuliert werden.

Σ_B muß für den Primärschlüssel von r_B eine entsprechende Integritätsbedingung enthalten. Dabei ergibt sich der Primärschlüssel als Vereinigung der Primärschlüssel der Relationsschemata, für welche die entsprechenden Entitätstypen gemeinsam einen Schlüssel des Beziehungstyps B darstellen (vgl. 2.1.11 und 2.2.1).