

Einige Dinge über Datenbanksysteme

<http://www.richiwolsch.de.vu>

2008

Zusammenfassung

Dieses kleine Pamphlet befasst sich mit einigen grundlegenden Dingen rund um Datenbanksysteme. In erster Linie ist es mir selbst eine Hilfe beim Lernen für die Klausur. Wie immer ist es aber natürlich auf meiner Homepage für jeden abrufbar. Und deshalb gilt: Wer's liest ist selber Schuld. Und wer's nicht liest auch! Mit sämtlichen Fehlern die du findest darfst du mir *bitte* meinen E-Mail-Postkasten zumüllen. Vielen Dank und frohe Ostern! Oder Weihnachten!

rw

Inhalt

Das Entity-Relationship-Modell	2
1 Reine Theorie	2
2 Funktionlität	2
3 Entity-Relationship-Diagramme	3
Das Relationenmodell	5
4 Noch ein Modell	5
5 Vom ER-Modell zum Relationenmodell	5
6 Relationenalgebra	6
7 Der relationale Verbund (Join)	9
8 Weil Beispiele so schön sind	10
Anfragesprachen	12
9 Datendefinitionssprache	12
10 Datenmanipulationssprache	13
11 Anfragen schachteln	15
12 Und wieder Beispiele	16
Entwurfstheorie	18
13 Funktionale Abhängigkeiten	18
14 Armstrong-Axiome und kanonische Überdeckung	19
15 Anomalien	21
16 Normalformen	22

Das Entity-Relationship-Modell

1 Reine Theorie

Am Anfang eines guten Projekts steht immer eine durchdachte Planung. Bevor man Datenbanken implementiert modelliert man sie. Das spart Arbeit, Zeit und Kosten. Denn eine Faustregel sagt, dass der Schaden der durch einen Fehler verursacht wird mit jeder Entwicklungsstufe exponentiell steigt. Das vermutlich verbreitetste Verfahren zur Modellierung von Datenbanken ist das *Entity-Relationship-Modell*, kurz ER-Modell. – Wie der Name schon sagt werden Entitäten (Dinge) und die zwischen ihnen bestehenden Beziehungen betrachtet. Einige Beispiele für solche Beziehungstypen sind

Entität	Entität	Beziehung
Kind	Vater	Vaterschaft
Kind	Eltern	Familienzugehörigkeit
Student	Fakultät	Immatrikulation
Stadt	Land	Hauptstadt
Flughafen	Flughafen	Linienflug
natürliche Zahl	natürliche Zahl	größer als

2 Funktionlütät

Ähnlich den Multiplizitäten der Unified Modeling Language (UML) können den Beziehungen auch im ER-Modell Wertigkeiten zugeordnet werden. Man nennt sie *Funktionalitäten*. Es gibt verschiedene Typen von Funktionalitäten.

- 1:1 – Jeder Entität, also jedem Ding wird durch die Beziehung genau ein (anderes) Ding zugeordnet.
- 1:N – Jedem Ding werden durch den Beziehungstypen beliebig viele andere Dinge zugeordnet.
- N:1 – analog 1:N
- N:M – Beliebigen Entitäten werden durch einen Beziehungstyp beliebig viele Entitäten zugeordnet.

Für die Beziehungstypen aus oben genanntem Beispiel gelten folgende Funktionalitäten:

Entität	Entität	Beziehung	Funktionalität
Kind	Vater	Vaterschaft	N:1
Kind	Eltern	Familienzugehörigkeit	N:1
Student	Fakultät	Immatrikulation	N:1
Stadt	Land	Hauptstadt	1:1
Flughafen	Flughafen	Linienflug	N:M
natürliche Zahl	natürliche Zahl	größer als	N:M

Natürlich kann man sich in einigen Fällen über die Funktionalitäten streiten. Zum Beispiel gehe ich davon aus, dass jeder Student nur an einer Fakultät studiert. Da gibt es aber auch Ausnahmefälle. Dem Beziehungstyp Immatrikulation könnte man also auch eine N:M-Funktionalität zuweisen. Im Falle des Beziehungstyps Familienzugehörigkeit sind hier nur die leiblichen Eltern berücksichtigt. Also hat ein Kind genau ein Elternpaar, aber jedes Elternpaar kann beliebig viele Kinder haben. Würde man auch Stiefeltern berücksichtigen wäre Familienzugehörigkeit eine N:M-Beziehung. Jedes Kind hätte dann beliebig viele Eltern.

Daraus ergibt sich aber ein Problem. Denn beliebig viele bedeutet auch keine. Man kann aber mit an Sicherheit grenzender Wahrscheinlichkeit sagen, dass es kein Kind ohne Eltern gibt. Deshalb gibt es eine weitere, genauere Möglichkeit Funktionalitäten aufzuschreiben. Diese Variante bezeichnet man als *Min-Max-Notation*. Ein Zahlentupel (*min*, *max*) gibt dabei den kleinsten und den größten Wert an. Ein Asterisk (*) steht dabei für eine beliebige Zahl. Zu beachten ist, dass die Notation genau andersherum zu den einfachen Funktionalitäten erfolgt! Im Folgenden sind den Funktionalitäten noch die Min-Max-Notationen hinzugefügt.

Entität	Entität	Beziehung	Funktionalität
Kind	Vater	Vaterschaft	N:1 (1,1):(1,*)
Kind	Eltern	Familienzugehörigkeit	N:1 (1,1):(1,*)
Student	Fakultät	Immatrikulation	N:1 (1,1):(1,*)
Stadt	Land	Hauptstadt	1:1 (0,1):(1,1)
Flughafen	Flughafen	Linienflug	N:M (1,:):(1,*)
natürliche Zahl	natürliche Zahl	größer als	N:M (1,:):(0,*)

Würde man bei der Beziehung Familienzugehörigkeit auch die Stiefeltern mit einbeziehen, dann hätte jedes Kind mindestens ein Elternpaar. Eine Höchstgrenze gibt es nicht. Jedes Elternpaar hat mindesten ein Kind, sonst ist es kein Elternpaar. Es kann aber auch mehr als ein Kind haben. Für die Beziehung Kind - Eltern gilt dann also: (1,:):(1,*).

3 Entity-Relationship-Diagramme

Solche Beziehungen lassen sich viel übersichtlicher in Diagrammen darstellen. Die vier gebräuchlichsten Elemente sind Entitytypen, Beziehungstypen, Attribute und Assoziationen (siehe Abb. 1). Ein Entitytyp kann dabei durch Assoziationen mit mehreren Beziehungstypen verbunden sein. Außerdem werden jedem Entitytyp Attribute zugeordnet. Anhand dieser Attribute kann ein bestimmtes Ding, also eine Entität genau identifiziert werden. Das Attribut oder die Attribute die zur eindeutigen Identifikation einer Entität mindestens notwendig sind nennt man Schlüsselattribute. Sie werden unterstrichen.

Die Funktionalitäten werden direkt an die Assoziationslinien geschrieben. Einige der Beziehungen aus dem Beispiel von oben könnten dann wie in Abb. 2 aussehen. Man beachte dabei, dass die Min-Max-Notation dabei umgekehrt zu den einfachen Funktionalitäten eingetragen wird.

Bei den Entitytypen Vater und Kind fällt ins Auge, dass es hier mehrere Schlüsselattribute gibt. Das ist der Fall, da die Person durch eines der beiden Attribute nicht eindeutig definiert wäre.

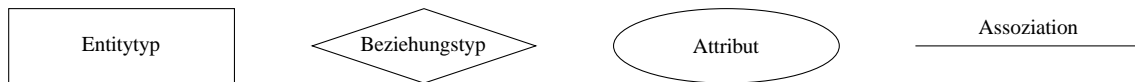


Abbildung 1: Elemente eines ER-Diagramms

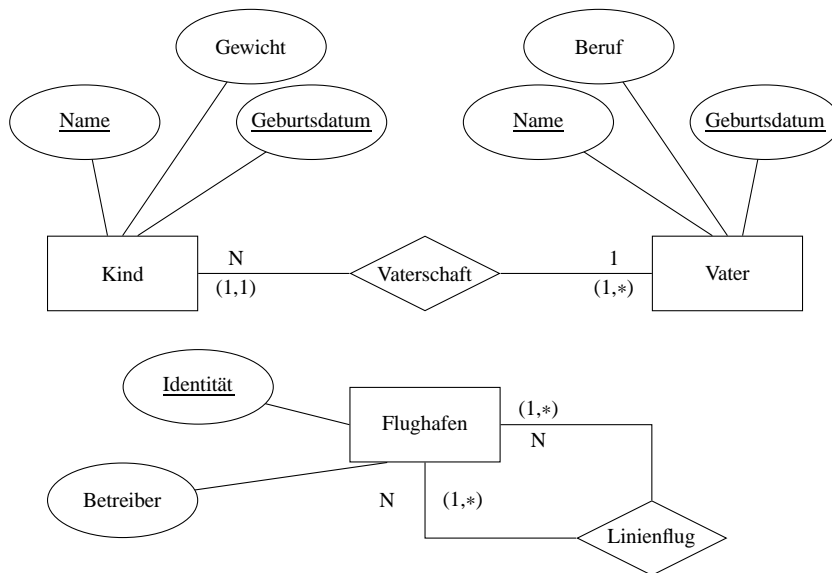


Abbildung 2: Die Beziehungen Vaterschaft und Linienflug im ER-Diagramm

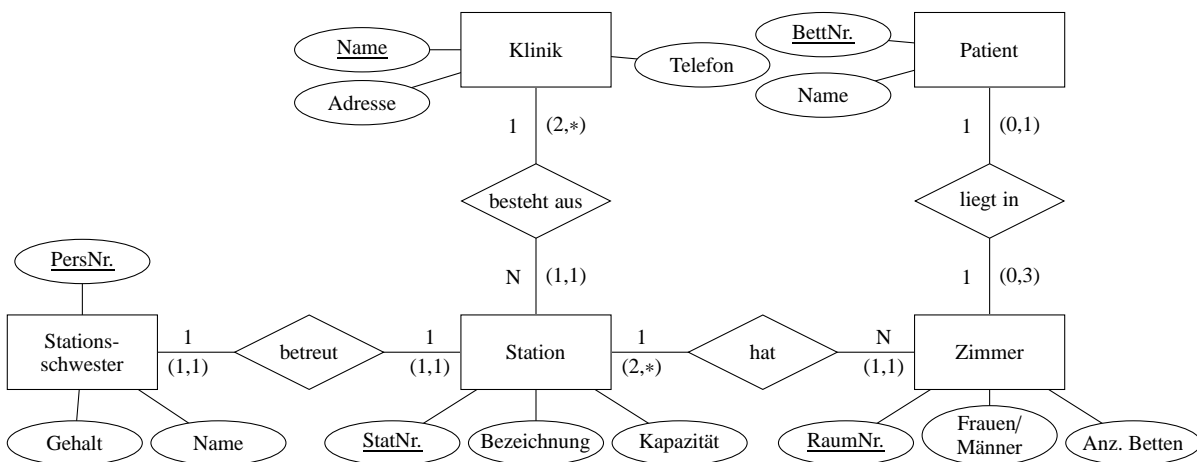


Abbildung 3: ER-Modell eines Krankenhauses

Das etwas komplexere Entity-Relationship-Diagramm in Abb. 3 modelliert einen Ausschnitt eines Krankenhauses. Danach besteht eine Klinik aus Stationen, eine Station hat mehrere Zimmer. Patienten werden ambulant oder stationär behandelt. Die Zimmer sind entweder Ein- oder Dreibettzimmer. In einem Zimmer dürfen entweder nur Frauen oder nur Männer untergebracht werden.

Wenn man ein Datenbanksystem als Bestandteil eines Softwareprojektes modellieren möchte bietet es sich natürlich an das gesamte Projekt einheitlich zu modellieren. Dazu ist die *UML* (Unified Modelling Language) sehr gut geeignet.

Das Relationenmodell

4 Noch ein Modell

Großer Beliebtheit erfreut sich auch das Relationenmodell, auch als *relationales Datenmodell* bezeichnet. Das liegt vermutlich daran, dass es sehr einfach aufgebaut ist. Im Prinzip nutzt das Modell einfache Tabellen zur Darstellung.

Jedes Relationenschema lässt sich in einer Tabelle darstellen. Ein Relationenschema ist eine Menge von Attributen. Zum Beispiel hat die Stationsschwester das Relationenschema (PersNr., Name, Gehalt). In anderer Literatur findet man auch die Schreibweise {[PersNr., Name, Gehalt]}. Dabei symbolisieren die Mengenklammern die Attributmenge und die eckigen Klammern das Attributtupel. Statt Entitytypen und Beziehungstypen kennt das relationale Datenmodell Relationen. Jede Relation ist auf einem Relationenschema definiert. Ein Attribut hat einen festgelegten Wertebereich. Dieser Wertebereich wird als *Domain* bezeichnet. Eine Tabelle stellt eine Instanz, bzw. Ausprägung eines Relationenschemas dar. Am besten wird das aber an einem Beispiel klar...

5 Vom ER-Modell zum Relationenmodell

Betrachten wir einmal das Börsengeschehen in Deutschland: Wertpapiere (z.B. Aktien, Rentenpapiere oder Investmentfonds) werden an verschiedenen Wertpapierbörsen gehandelt. Nicht alle Wertpapiere werden an einer Börse gehandelt. Nehmen wir aber an, an jeder Wertpapierbörse würden mindestens 50 Wertpapiere gehandelt. Wertpapiere werden durch eine eindeutige Kennnummer bezeichnet. Abbildung 4 zeigt das ER-Diagramm dieses Szenarios.

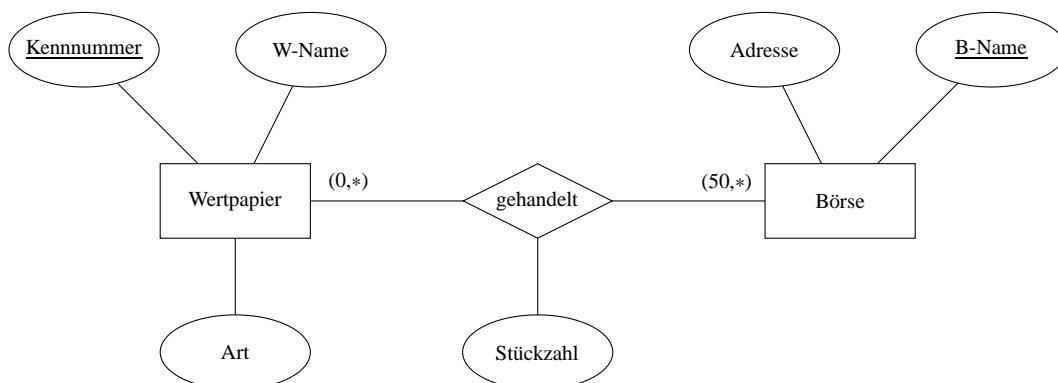


Abbildung 4: Entity-Relationship-Modell des Wertpapierhandels

Im Relationenmodell stellt sich das wie folgt dar:

Wertpapier(Kennnummer, W-Name, Art)
 Börse(B-Name, Adresse)
 gehandelt(Kennnummer, B-Name, Stückzahl)

Die Relation gehandelt hat einen Primärschlüssel aus zwei Attributen, da sie nur durch die Kombination der Attribute Kennnummer und B-Name eindeutig identifiziert ist. Ein Primärschlüssel muss immer drei Bedingungen genügen:

1. Eindeutigkeit: Der Primärschlüssel darf in jeder Relation nur einmal vorkommen, d.h. er muss eine bestimmte Ausprägung der Relation identifizieren.
2. Definiertheit: Der Primärschlüssel muss für jede Ausprägung der Relation definiert sein.
3. Minimalität: Dem Primärschlüssel dürfen nur so viele Attribute angehören, wie maximal nötig, um Eindeutigkeit und Definiertheit zu gewährleisten.

Unter der Ausprägung einer Relation versteht man die Werte die die Attributtupel annehmen können. Eine Ausprägung von Börse ist zum Beispiel (Frankfurter Börse, 60485 Frankfurt am Main)

Im obigen Beispiel enthält die Relation gehandelt auch zwei Fremdschlüssel, die den Bezug zu anderen Relationen herstellen. Dabei ist Kennnummer der Fremdschlüssel für das Wertpapier und B-Name ist der Fremdschlüssel für die Börse, an der das Wertpapier gehandelt wird. Man kann dies so kennzeichnen:

gehandelt(Kennnummer → Wertpapier, B-Name → Börse, Stückzahl)

6 Relationenalgebra

Irgendwie möchte man nun aus dem Datenbanksystem Informationen extrahieren. Das ist unter anderem mit der *relationalen Algebra* (Relationenalgebra) möglich. Damit lässt sich leicht ein Ablaufplan beschreiben, wie Relationen auszuwerten sind. Als Ergebnis erhält man wieder Relationen. Die Grundoperationen der Relationenalgebra sind Selektion, Projektion, Vereinigung, Mengendifferenz, Kartesisches Produkt und Umbenennung.

Um die Operationen zu veranschaulichen nehmen wir nocheinmal den Beziehungstyp Vaterschaft her, der bis auf wenige Änderungen diesem relationalen Modell entspricht:

Kind(KName, Alter, Gewicht)
 Vater(VName, Alter, Gewicht)
 Vaterschaft(KName, VName, leiblich)

Es könnte beispielsweise diese Ausprägung haben:

Kind	Name	Alter	Gewicht
	Alf Müller	10	50
	Marcell Koch	16	65
	Marie Lange	16	58
	Victor Lange	8	40
	Nadine Mayer	8	40

Vater	Name	Alter	Gewicht
	Uwe Müller	42	91
	Rolf Koch	36	78
	Reinhold Lange	43	75
	Hugo Mayer	27	70

Vaterschaft	KName	VName	leiblich
	Alf Müller	Uwe Müller	ja
	Marcell Koch	Rolf Koch	ja
	Marie Lange	Reinhold Lange	ja
	Victor Lange	Reinhold Lange	nein
	Nadine Mayer	Hugo Mayer	ja

Selektion: Die Selektion $\sigma_F(R)$ bestimmt alle Tupel einer Relation R , die einer Bedingung (Formel) F genügen. Die Formel kann aus folgenden Komponenten zusammengesetzt sein: Operanden (Attributnamen oder Konstanten), Vergleichsoperatoren ($=$, $<$, \leq , $>$, \geq , \neq) und booleschen Operatoren (\wedge , \vee , \neg). Die Selektion wählt also einzelne Zeilen aus der Relation aus.

Beispiel: alle Kinder die älter als 14 Jahre sind

$\sigma_{\text{Alter} > 14}(\text{Kind})$	Name	Alter	Gewicht
	Marcell Koch	16	65
	Marie Lange	16	58

Projektion: Die Projektion $\pi_A(R)$ bestimmt alle in einer Ausprägung vorkommenden Werte der Attribute A . Die Attribute werden einfach durch Komma getrennt aufgelistet. Die Projektion wählt also einzelne Spalten aus der Relation aus. Dabei werden Duplikate eliminiert.

Beispiel: alle Altersstufen der Kinder

$\pi_{\text{Alter}}(\text{Kind})$	Alter
	10
	16
	8

Vereinigung: Die Vereinigung zweier Relationen $R_1 \cup R_2$ ist nur möglich, wenn beide das gleiche Schema (gleiche Attributnamen und Attributtypen) aufweisen. Die Attribute können dabei wenn nötig umbenannt werden. Aus der neuen Relation sind die Duplikate zu eliminieren.

Beispiel: Name, Alter und Gewicht aller Kinder und Väter

Kind \cup Vater	Name	Alter	Gewicht
	Alf Müller	10	50
	Marcell Koch	16	65
	Marie Lange	16	58
	Victor Lange	8	40
	Nadine Mayer	8	40
	Uwe Müller	42	91
	Rolf Koch	36	78
	Reinhold Lange	43	75
	Hugo Mayer	27	70

Mengendifferenz: Die Mengendifferenz $R_1 - R_2$ kann nur gebildet werden, wenn beide Relationen das gleiche Schema aufweisen. Die Ergebnisrelation enthält dann alle Tupel, die in R_1 , aber nicht in R_2 vorkommen.

Beispiel: Namen aller Adoptivkinder

$\pi_{\text{Name}}(\text{Kind}) - \pi_{\text{KName}}(\sigma_{\text{leiblich} = \text{ja}}(\text{Vaterschaft}))$	Name
	Victor Lange

Natürlich bedürfte es nicht dieses komplizierten Konstrukts, um die Adoptivkinder zu ermitteln. Aber es ist eben nur ein Beispiel.

Kartesisches Produkt: Das kartesische Produkt (Kreuzprodukt) $R_1 \times R_2$ enthält alle möglichen Kombinationen der Tupel aus R_1 und R_2 . Die neue Relation enthält also $|R_1| \cdot |R_2|$ Tupel. Durch gleichbenannte Attribute kann es notwendig sein Attribute umzubenennen. Eine häufig genutzte Möglichkeit ist es dem Attributnamen die Relation und einen Punkt voranzustellen, z.B. Kind.Alter und Vater.Alter.

Beispiel: alle Kombinationen aus Vater und Kind (\rightarrow 20 Tupel)

Vater \times Kind	V.Name	V.Alt	V.Gew	K.Name	K.Alt	K.Gew
	Uwe Müller	42	91	Alf Müller	10	50
	Uwe Müller	42	91	Marcell Koch	16	65

	Hugo Mayer	27	70	Victor Lange	8	40
	Hugo Mayer	27	70	Nadine Mayer	8	40

Umbenennung: Es können sowohl Relationen als auch Attribute umbenannt werden: $\varphi_{R_2}(R_1)$ und $\varphi_{A_2 \leftarrow A_1}(R)$. Umbenennungen von Relationen sind notwendig, wenn eine Relation mehrfach in einer Anfrage vorkommt. Falsch wäre $R \times R$. Stattdessen könnte man aber $R \times \varphi_S(R)$ schreiben. Die Notwendigkeit zur Umbenennung von Attributen kann sich durch den Joinoperator (Seite 9) ergeben.

Um einige gebräuchliche Anfragen unkompliziert ausdrücken zu können, kennt die relationale Algebra weitere abgeleitete Operatoren, wie den Mengendurchschnitt, die relationale Division und den relationalen Verbund (Join).

Mengendurchschnitt: Der Mengendurchschnitt $R_1 \cap R_2$ wählt alle Tupel aus, die in beiden Relationen vorkommen. Die Relationen müssen nach dem gleichen Schema sein. Es ist $R_1 \cap R_2 = R_1 - (R_1 - R_2)$. Beispiel: Namen aller Väter die ein Adoptivkind haben und älter als 40 Jahre sind

$\pi_{\text{Name}}(\sigma_{\text{leiblich}=\text{nein}}(\text{Vaterschaft})) \cap \pi_{\text{Name}}(\sigma_{\text{Alter}>40}(\text{Vater}))$	Name
	Reinhold Lange

Relationale Division: Die relationale Division $R_1 \div R_2$ bietet die Möglichkeit nur Tupel auszuwählen, für die „alle Bedingungen erfüllt sind“, d.h. Anfragen mit Allquantor auszuführen. Am deutlichsten wird das wohl an einem Beispiel: Gesucht seien die Touristen, die bereits *alle* Städte mit mehr als 5 Mio. Einwohnern besucht haben. Gegeben ist die folgende Datenbank:

Tourist(Name, Alter, Heimatstadt)
 Stadt(Ortsname, Einwohner, Land)
 besucht(Name \rightarrow Tourist, Ortsname \rightarrow Stadt)

Es werden zuerst die Namen aller Städte mit mehr als 5 Mio. Einwohnern ermittelt.

$$\text{Metropolen} := \pi_{\text{Ortsname}}(\sigma_{\text{Einwohner}>5\text{Mio}}(\text{Stadt}))$$

Die relationale Division übernimmt nun die Aufgabe zu überprüfen, welche Touristen in allen diesen Städten waren. Dabei wird jedes Tupel aus der einen mit jedem Tupel aus der anderen Relation verglichen. Es wird überprüft, ob die Attribute in Metropolen die selben Werte haben wie die gleichnamigen Attribute in besucht. Deshalb darf die zweite Relation immer nur Attribute enthalten, die in der ersten auch vorkommen. Sind die Werte der Attribute gleich, so wird der „Rest“ des Tupels aus der ersten Relation (hier: besucht) in die Ergebnisrelation eingefügt. Diese enthält dann automatisch nur die Namen der Touristen.

$$\text{Großstadttouristen} := \text{besucht} \div \text{Metropolen}$$

Die relationale Division setzt sich aus diesen Grundoperationen zusammen:

$R_1 \div R_2 = \pi_{A_1-A_2}(R_1) - \pi_{A_1-A_2}((\pi_{A_1-A_2}(R_1) \times R_2) - R_1)$ wobei A_1 die Attributmenge von R_1 und A_2 die von R_2 ist.

7 Der relationale Verbund (Join)

Der Join ist ein weiterer abgeleiteter Operator der Relationenalgebra. Im Prinzip handelt es sich beim Join um eine Verbesserung des kartesischen Produkts. Dessen Problem ist die unangenehm große Ergebnisrelation, von der man oft nur wenige Tupel benötigt. Beim Join wird bereits eine Vorauswahl getroffen. Einige Joins sind Allgemeiner Verbund, Natürlicher Verbund, Halbverbund, sowie linker, rechter und vollständiger äußerer Verbund.

Allgemeiner Verbund: Beim allgemeinen Verbund $R_1 \bowtie_{\theta} R_2$ wird die Bedingung für die Vorauswahl als Formel θ übergeben. Die Formel kann die folgenden logischen Operationen enthalten: $=, \neq, <, \leq, \geq, >$. Wenn die Formel nur Gleichheitszeichen enthält, nennt man das einen Equi-Join. Der Theta-Join ist zusammengesetzt aus einem Kreuzprodukt und einer nachfolgenden Selektion.

Beispiel: Bleiben wir bei den Touristen. Wir wollen jedem Touristen sein Heimatland zuordnen. Zuerst wird jedem die Heimatstadt zugeordnet.

$$\text{Tourist} \bowtie_{\text{Heimatstadt} = \text{Ortsname}} \text{Stadt}$$

Nun kann noch eine Projektion auf den Namen des Touristen und das Land erfolgen.

$$\pi_{\text{Name}, \text{Land}}(\text{Tourist} \bowtie_{\text{Heimatstadt} = \text{Ortsname}} \text{Stadt})$$

Natürlicher Verbund: Ganz allgemein kann man sagen, dass der natürliche Join $R_1 \bowtie R_2$ die intuitiv zusammengehörigen Tupel zweier Relationen zu einem Tupel vereint. Dazu wird wieder das Kreuzprodukt gebildet. Es werden aber nur die Tupel zurückgegeben, bei denen Attribute mit dem selben Namen die gleichen Werte haben. Die Ergebnisrelation enthält dieses Attribut natürlich nur einmal, nicht doppelt. Unvollständige Tupel, die nur in einer Argumentrelation auftauchen werden weggeschmissen. Da nur Attribute mit dem selben Namen verglichen werden (nicht aber Attribute, mit denen daselbe gemeint ist) müssen Attribute vorher häufig umbenannt werden. Das ist aber mit dem φ -Operator leicht möglich. Beispiel: Wir wollen alle vorhandenen Informationen über Väter und ihre Kinder zusammenbringen. Dazu kombinieren wir zuerst die Relationen Vater und Vaterschaft und erhalten das folgende Relationenschema:

(KName, VName, leiblich, Alter, Gewicht)

$$\text{Vaterschaft} \bowtie (\varphi_{\text{VName} \leftarrow \text{Name}}(\text{Vater}))$$

Um eine Vergleichsbasis zu schaffen muss dazu das Attribut Name in Vater in VName umbenannt werden. Um nun auch noch die Kind-Informationen hinzuzufügen müssen die Attribute Alter und Gewicht noch umbenannt werden. Das kann vor oder nach dem Join erfolgen.

$$\text{Vaterschaft} \bowtie (\varphi_{\text{VName} \leftarrow \text{Name}, \text{VAlter} \leftarrow \text{Alter}, \text{VGewicht} \leftarrow \text{Gewicht}}(\text{Vater}))$$

Zum Schluss noch die Kinder:

$$\begin{aligned} V & := \text{Vaterschaft} \bowtie (\varphi_{\text{VName} \leftarrow \text{Name}, \text{VAlter} \leftarrow \text{Alter}, \text{VGewicht} \leftarrow \text{Gewicht}}(\text{Vater})) \\ & \quad \bowtie (\varphi_{\text{KName} \leftarrow \text{Name}, \text{KAlter} \leftarrow \text{Alter}, \text{KGewicht} \leftarrow \text{Gewicht}}(\text{Kind})) \end{aligned}$$

Da der natürliche Join kommutativ und assoziativ ist kann einfach ein weiterer Join hinten angehängt werden. Es ergibt sich die folgende Relation:

V	KName	VName	leibl.	VAlter	VGewicht	KAlter	KGewicht
	Alf Müller	Uwe Müller	ja	42	91	10	50
	Marcell Koch	Rolf Koch	ja	36	78	16	65
	Marie Lange	Reinhold Lange	ja	43	75	16	58
	Victor Lange	Reinhold Lange	nein	43	75	8	40
	Nadine Mayer	Hugo Mayer	ja	27	70	8	40

Äußerer Join: Die vorangegangenen Operatoren werden auch als innere Joins bezeichnet. Beim äußeren Join bleiben zusätzlich zu den gefilterten Tupeln die restlichen Tupel einer oder beider Relationen erhalten:

- Der linke äußere Join $R_1 \bowtie R_2$ erhält die Tupel von R_1 .
- Der rechte äußere Join $R_1 \ltimes R_2$ erhält die Tupel von R_2 .
- Der vollständige äußere Join $R_1 \ltimes R_2$ erhält die Tupel beider Argumentrelationen.

Die leeren Felder werden mit Null-Werten aufgefüllt.

Halbverbund: Der Halbverbund oder Semi-Join $R_1 \ltimes R_2$ liefert nur die Tupel aus R_1 zurück, die die Join-Bedingung mit R_2 erfüllen. Die Ergebnisrelation enthält also alle Tupel aus R_1 , die einen potentiellen Join-Partner in R_2 haben.

8 Weil Beispiele so schön sind

Der Fußballverband führt für eine Saison folgende Datenbank:

Verein (VName, Ort, Präsident)
 Spiele (Heim, Gast, Resultat, Zuschauer, Termin, Spieltag, HTrainer, GTrainer)
 Spieler (SNr, Name, VName, Alter, Gehalt, GebOrt)
 Trainer (TNr, Name, VName, Alter, Gehalt, GebOrt)
 Einsatz (Heim, Gast, SNr, von, bis, Tore, Karte)

1. Welche Spiele sind 2:0 ausgegangen?
 $\sigma_{\text{Resultat} = 2:0}(\text{Spiele})$
2. Welche Spieler (Name) kamen beim Spiel Olympia Bremen - 1.FC Oelsnitz zum Einsatz?
 $\pi_{\text{Name}}((\sigma_{\text{Heim} = \text{Olympia Bremen} \wedge \text{Gast} = \text{1.FC Oelsnitz}}(\text{Einsatz})) \bowtie \text{Spieler})$
3. Welche Spieler (Name) spielen in einem Verein ihres Geburtsorts?
 $\pi_{\text{Name}}(\text{Spieler} \bowtie_{\text{GebOrt} = \text{Ort} \wedge \text{Spieler.VName} = \text{Verein.VName}} \text{Verein})$ oder
 $\pi_{\text{Name}}(\sigma_{\text{GebOrt} = \text{Ort}}(\text{Spieler} \bowtie \text{Verein}))$
4. Welche Spieler (SNr) haben bisher noch kein Tor geschossen?
 $\pi_{\text{SNr}}(\text{Spieler}) - \pi_{\text{SNr}}(\sigma_{\text{Tore} \neq 0}(\text{Einsatz}))$
5. Welche Spieler (Name) haben ein höheres Gehalt als der Trainer?
 $\pi_{\text{Spieler.Name}}(\text{Trainer} \bowtie_{\text{Trainer.Gehalt} < \text{Spieler.Gehalt} \wedge \text{Trainer.VName} = \text{Spieler.VName}} \text{Spieler})$ oder
 $\pi_{\text{Spieler.Name}}(\sigma_{\text{Spieler.Gehalt} > \text{Trainer.Gehalt}}(\text{Spieler} \bowtie_{\text{Spieler.VName} = \text{Trainer.VName}} \text{Trainer}))$
6. Wie heißen die Präsidenten der Vereine, die zur Zeit einen Trainer beschäftigen, der jünger ist als der älteste Spieler?
 $\pi_{\text{Präsident}}(\text{Verein} \bowtie (\varphi_{\text{VName} \leftarrow \text{Trainer.VName}}(\text{Trainer} \bowtie_{\text{Trainer.VName} = \text{Spieler.VName} \wedge \text{Trainer.Alter} < \text{Spieler.Alter}} \text{Spieler})))$
7. Welche Spieler (SNr, Name) haben bisher noch nie gespielt?
 $\pi_{\text{SNr, Name}}(\text{Spieler} - (\text{Spieler} \ltimes \text{Einsatz}))$ oder
 $\pi_{\text{SNr, Name}}(\text{Spieler} \bowtie (\pi_{\text{SNr}}(\text{Spieler}) - \pi_{\text{SNr}}(\text{Einsatz})))$

8. Welche Spieler (SNr), die 20 Jahre und jünger sind, wurden schon eingesetzt?
 $\pi_{\text{SNr}}(\text{Einsatz} \bowtie (\sigma_{\text{Alter} \leq 20}(\text{Spieler})))$
9. In welchen Orten gab es bereits Spiele mit mehr als 10 000 Zuschauern?
 $\pi_{\text{Ort}}(\text{Verein} \bowtie_{\text{VName=Heim}} (\sigma_{\text{Zuschauer} > 10000}(\text{Spiele})))$
10. Welche Spieler (SNr) sind bei allen Spielen von Aufbau Borgheide zum Einsatz gekommen?
 $\pi_{\text{SNr}}((\sigma_{\text{VName=Aufbau Borgheide}}(\text{Spieler})) \div (\pi_{\text{SNr}}(\sigma_{\text{Heim=Aufbau Borgheide} \vee \text{Gast=Aufbau Borgheide}}(\text{Einsatz}))))$
11. Welche Gehälter haben die Trainer und Spieler (Name)?
 $(\pi_{\text{Name, Gehalt}}(\text{Trainer})) \cup (\pi_{\text{Name, Gehalt}}(\text{Spieler}))$
12. In der Datenbank befinden sich auch Vereine, deren Trainer nicht bekannt ist. Welche Vereine (VName) werden von wem trainiert (Name)? (Die Ergebnisrelation soll alle Vereine enthalten.)
 $\pi_{\text{VName, Name}}(\text{Verein} \bowtie \text{Trainer})$
-

Anfragesprachen

9 Datendefinitionssprache

Jetzt wird es praktisch. Ein bisschen jedenfalls. Die Anfragesprachen dienen der Umsetzung der Relationenalgebra in den verschiedenen Datenbanksystemen. Datenbanken kennen grundsätzlich zwei Sprachen. Es gibt die *Datendefinitionssprache* (data definition language, DDL) und *Datenmanipulationssprache* (data manipulation language, DML). Am weitesten verbreitet ist die *Structured Query Language* (SQL). Sie umfasst die Definition und Manipulation von Datenbanken.

Die Datendefinitionssprache erlaubt es das Datenbankschema festzulegen. Im Allgemeinen läuft das darauf hinaus, dass Tabellenköpfe angelegt werden. Den einzelnen Attributen müssen Datentypen zugeordnet werden. Die wichtigsten ANSI-genormten Datentypen sind:

- `char(n)` Zeichenkette mit der Länge `n` (wird ggf. mit Leerzeichen aufgefüllt)
- `varchar(n)` Zeichenkette mit der maximalen Länge `n` (character varying)
- `numeric(p,s)` Zahl mit `p` Stellen und `s` Nachkommastellen, außerdem gibt es `integer` und `float`
- `date` Datum
- `blob` (binary large object), bzw. `raw` für Binärdaten
- `xml` XML-formatierte Daten

Ein Relationenschema lässt sich mit dem Befehl `create table` anlegen.

```
create table Stadt
(Name varchar(20) not null,
 Kreis varchar(30),
 Einwohner integer);
```

Mit dem Zusatz `not null` wird gewährleistet, dass das Attribut immer definiert ist. Das ist notwendig, wenn das Attribut als Primärschlüssel genutzt werden soll. Der SQL-Befehl erzeugt das folgende Relationenschema:

```
Stadt(Name, Kreis, Einwohner)
```

Zum Ändern und Löschen des Schemas stehen die Befehle `alter table` und `drop table` zur Verfügung. Die Datenbank wird folgendermaßen mit Tupeln gefüllt:

```
insert into Stadt
(Plauen, Vogtlandkreis, 68000);
```

10 Datenmanipulationssprache

Um einige Anfragen beispielhaft erklären zu können sei die folgende Datenbank mit geografischen Daten über das Bundesland Sachsen definiert:

```
Stadt(Name, Kreis, Einwohner)
Kreis(Name, Kreisstadt, Einwohner, Fläche, Direktion)
Berg(Name, Gebirge, Höhe)
Fluss(Name, Fluss, Meer, Länge)
benachbart(Kreis1, Kreis2)
liegtan(Stadt, Fluss)
```

Die Attribute Fluß und Meer im Relationenschema Fluß bezeichnen das Gewässer, in das ein Fluß mündet. Es ist also jeweils nur einer der beiden Werte belegt. Direktion bezeichnet die Landesdirektion (Regierungsbezirk). Die Datenbank kann als Microsoft Access-Datei auf meiner Homepage heruntergeladen werden. Im SQL-Anfragemodus lassen sich die einzelnen Beispiele dann nachvollziehen.

Um aus einer Tabelle die Werte einzelner Attribute auszuwählen verwendet man in SQL die Anweisungen `select` und `from`. Dies entspricht der Projektion der relationalen Algebra. Die Anweisung

```
select Name, Direktion
from Kreis;
```

liefert diese Ausgabe:

Name	Direktion
Bautzen	Dresden
Erzgebirgskreis	Chemnitz
Leipzig	Leipzig
Meißen	Dresden
Mittelsachsen	Chemnitz
Görlitz	Dresden
Nordsachsen	Leipzig
Sächsische Schweiz-Osterzgeb.	Dresden
Vogtlandkreis	Chemnitz
Zwickau	Chemnitz

Möchte man nun alle Landesdirektionen anzeigen muss man beachten, dass SQL die Duplikateliminierung nicht automatisch ausführt. Es reicht also nicht eine Projektion auf die Spalte Direktion durchzuführen. Zusätzlich müssen noch mittels des Zusatzes `distinct` alle Duplikate eliminiert werden.

```
select distinct Direktion
from Kreis;
```

Die umgekehrte Sortierung erreicht man mittels `desc` (descending). Aufsteigende Sortierung ist standardmäßig eingestellt, man kann sie aber auch mit `asc` (ascending) festlegen.

```
select distinct Direktion
from Kreis
order by Direktion desc;
```

Direktion
Leipzig
Dresden
Chemnitz

Um nur Kreise in der Landesdirektion Leipzig anzuzeigen kann die Anfrage mittels `where` weiter verfeinert werden.

```
select Name, Direktion
from Kreis
where Direktion='Leipzig';
```

liefert diese Ausgabe:

Name	Direktion
Leipzig	Leipzig
Nordsachsen	Leipzig

Die meisten SQL-Schlüsselwörter sind recht aussagekräftig, weshalb im Folgenden nur eine Auswahl erklärt ist.

Average: Mittels `avg` lässt sich der Durchschnitt bestimmen. Die durchschnittliche Fläche der sächsischen Landkreise ergibt sich aus

```
select avg(Fläche)
from Kreis;
```

und ist 1 756,9 (km^2).

Group by: Das Schlüsselwort `group by` erlaubt es Werte zu gruppieren. Soll die durchschnittliche Fläche der Landkreise nach Landesdirektionen getrennt ermittelt werden, so ließe sich das mit folgender Anfrage bewerkstelligen:

```
select Direktion, avg(Fläche)
from Kreis
group by Direktion;
```

Direktion	avg(Fläche)
Chemnitz	1 575
Dresden	1 900,75
Leipzig	1 833

Having: Ähnlich dem `where`-Schlüsselwort lässt sich mittels `having` eine Bedingung an die `group by`-Anweisung knüpfen. Die durchschnittliche Größe der Kreise in Direktionen, die im Schnitt mehr als 250 000 Einwohner haben, ergibt sich wie folgt:

```
select Direktion, avg(Fläche), avg(Einwohner)
from Kreis
group by Direktion
having avg(Einwohner) > 250000;
```

Direktion	avg(Fläche)	avg(Einwohner)
Chemnitz	1 575	336 500
Dresden	1 900,75	288 000

Zusätzlich wird hier die durchschnittliche Einwohnerzahl mit ausgegeben, die in beiden Direktionen entsprechend über 250 000 liegt.

Andere Schlüsselwörter sind `and`, `or`, `max`, `min`, `sum` und `count`.

11 Anfragen schachteln

Viel mächtiger wird dieses Werkzeug, wenn man Anfragen ineinander verschachtelt. Nun könnte man alle Kreise ausfindig machen, die unterdurchschnittlich viele Einwohner haben.

```
select Name, Einwohner
from Kreis
where Einwohner < (select avg(Einwohner)
                   from Kreis);
```

Das sind immerhin sechs aller zehn Landkreise:

Name	Einwohner
Leipzig	277000
Meißen	261000
Görlitz	293000
Nordsachsen	217000
Sächsische Schweiz-Osterzgeb.	260000
Vogtlandkreis	257000

Um alle Städte zu ermitteln, die mehr Einwohner haben, als der kleinste Landkreis gibt es verschiedene Möglichkeiten:

```
select *
from Stadt
where Einwohner > (select min(Einwohner)
                   from Kreis);
```

Der Stern (*) gibt an, dass alle Attribute in die Ergebnistabelle übernommen werden. Es erfolgt also keine Projektion.

Eine andere Möglichkeit ist es nun den Operator `exists` zu nutzen. Der gibt nur dann `true` zurück, wenn die ihm untergeordnete Anfrage mindestens ein Ergebnistupel zurückliefert. Damit könnte man in einer Unteranfrage überprüfen, ob die soeben ausgewählte Stadt mehr Einwohner hat als ein Kreis. Wenn diese Unteranfrage mindestens ein Tupel enthält wird die Stadt zurückgegeben.

```
select *
from Stadt s
where exists (select *
              from Kreis k
              where k.Einwohner < s.Einwohner);
```

Um in der Unteranfrage auf beide Tabellen (Stadt und Kreis) zugreifen zu können muss ihnen ein Name zugewiesen werden. Die Stadt wird hier mit dem Kürzel `s`, der Kreis mit dem Kürzel `k` bezeichnet. Im Prinzip sind solche Benennungen immer möglich und manchmal auch ganz sinnvoll, um Verwechslungen zwischen Attributen mehrerer Tabellen zu vermeiden.

Beide SQL-Anfragen führen zu folgender Ergebnistabelle:

Name	Kreis	Einwohner
Leipzig		506000
Dresden		505000
Chemnitz		246000

12 Und wieder Beispiele

Bleiben wir bei der Datenbank über Sachsen und klären noch einige Beispiele. Im folgenden beschreibe ich nur die SQL-Anweisungen zu den jeweiligen Anfragen, nicht die Ergebnistabellen, sonst wären wir schnell bei hundert Seiten. Wer Lust hat kann sich die Access-Datenbank herunterladen und die Anfragen selbst eingeben.

1. Wie lauten die Namen aller Landesdirektionen?

```
select Direktion                select distinct Direktion
from Kreis                      from Kreis;
group by Direktion;
```

2. Wie heißen alle Kreise (Name) in der Landesdirektion Dresden?

```
select Name
from Kreis
where Direktion = 'Dresden';
```

3. Wie heißen die Berge des Erzgebirges? Dabei ist zu beachten, dass das Erzgebirge in Teilgebirge aufgespalten ist. Zum Vergleich von ähnlichen Einträgen kann man den Befehl like und Wildcards verwenden.

```
select Name
from Berg
where Gebirge like '*erzgebirge*';
```

4. Wie heißen die Flüsse, die in die Elbe münden und über 100km lang sind?

```
select Name
from Fluss
where Fluss = 'Elbe' and Länge > 100;
```

5. Welche Gebirge haben Berge höher als 800m? Dabei ist zu beachten, dass nicht jeder Berg einem Gebirge zugeordnet ist.

```
select distinct Gebirge
from Berg
where Höhe > 800 and Gebirge is not null;
```

6. Wie lauten die Namen und die Einwohnerzahlen aller Städte mit mehr als 50 000 Einwohnern? Das Ergebnis soll absteigend nach Einwohnern sortiert sein.

```
select Name, Einwohner
from Stadt
where Einwohner > 50000
order by Einwohner desc;
```

7. Wie groß ist der Einwohneranteil der Kreisstädte an den Gesamteinwohnern der Kreise?

```
select sum(s.Einwohner)/sum(k.Einwohner)*100 as Prozent
from Kreis k, Stadt s
where k.Kreisstadt = s.Name;
```

Der Anteil wird hier in Prozent berechnet und in einer als „Prozent“ benannten Spalte ausgegeben.

8. Wie viele Städte liegen jeweils an den Flüssen? Die Flüsse sollen absteigend nach ihrer Länge geordnet sein.

```
select f.Name, f.Länge, count(l.Stadt) as AnzOrte
from Fluss f, liegtan l
where f.Name = l.Fluss
group by f.Name, f.Länge
order by f.Länge desc;
```

9. Welche Kreisstädte liegen an einem Fluss? Die Kreisstädte mit Einwohnerzahl und Name des Flusses sollen der Größe nach geordnet sein.

```
select s.Name, s.Einwohner, l.Fluss
from Kreis k, liegtan l, Stadt s
where k.Kreisstadt = l.Stadt and s.Name = k.Kreisstadt
order by s.Einwohner desc;
```

10. Wieviele Kreise grenzen sowohl an Mittelsachsen, als auch an den Kreis Bautzen?

```
select count(*) as AnzKreise
from (select *
      from benachbart
      where Kreis1 = 'Bautzen' or Kreis2 = 'Bautzen') tmp,
      benachbart b
where (b.Kreis1 = 'Mittelsachsen' and b.Kreis2
      in (tmp.Kreis1, tmp.Kreis2))
or (b.Kreis2 = 'Mittelsachsen' and b.Kreis1
      in (tmp.Kreis1, tmp.Kreis2));
```

Hier wird in der inneren Projektion zuerst eine Tabelle (tmp) gebildet, die nur Kreise enthält, die neben Bautzen liegen. Danach wird überprüft, welche dieser Kreise auch an Mittelsachsen grenzen. Diese werden gezählt.

11. Welches ist der höchste Berg Sachsens?

```
select *
from Berg
where Höhe = (select max(Höhe)
              from Berg);
```

12. Welche Kreise (Name) haben wieviele Städte mit mehr als 25 000 Einwohnern?

```
select Kreis, count(Name) as AnzStädte
from Stadt
where Einwohner > 25000 and Kreis is not null
group by Kreis;
```

Entwurfstheorie

13 Funktionale Abhängigkeiten

Ziel der rationalen Entwurfstheorie ist es Abhängigkeiten in einer Datenbank auszunutzen, um diese zu optimieren. Die Datenbank eines Versandhandels könnte zum Beispiel so aussehen:

Versand(Kundennr., Auftragsnr., Artikelnr., Name, Adresse, Ware, Preis)

Diese Datenbank enthält aber redundante Einträge. Beispielsweise müssen Adresse und Name eines Kunden nicht bei jedem Auftrag neu gespeichert werden. Sie sind *funktional abhängig* von der Kundennummer. Außerdem müsste eine Adressänderung bei jedem Eintrag ausgeführt werden, der die Kundennummer des betreffenden Kunden enthält um *Anomalien* in der Datenbank zu vermeiden. Die Zuordnung von Kundennummer, Name und Adresse muss konsistent sein. Besser wäre es also man würde die Datenbank aufteilen.

Kunde(Kundennr., Name, Adresse)

Auftrag(Kundennr., Auftragsnr., Artikelnr., Ware, Preis)

Ware und Preis sollte man nicht in ein weiteres Schema auslagern, da sich der Preis einer Ware ändern kann und der Preis damit von der Ware nicht funktional abhängig ist.

Funktionale Abhängigkeiten (*functional dependencies*, kurz FD) sind eine spezielle Art von Integritätsbedingungen. Das heißt sie stellen eine Bedingung an die Ausprägung eines Relationenschemas. Damit kann gewährleistet werden, dass alle Werte die die Datenbank enthält, bestimmten Regeln folgen müssen. Wenn die Adresse von der Kundennummer abhängig ist, dann können ein und derselben Kundennummer nicht zwei verschiedene Adressen zugeordnet werden. Man sagt auch die Kundennummer bestimmt die Adresse funktional: {Kundennummer} → {Adresse}.

Formal gilt:

$$A \rightarrow B \Leftrightarrow s, t \in R(\mathcal{R}) : s[A] = t[A] \Rightarrow s[B] = t[B]$$

A bestimmt B funktional, genau dann wenn die Werte $s[A]$ und $t[A]$ einer Attributmenge A zweier Ausprägungen s und t von \mathcal{R} übereinstimmen, dann stimmen auch die Werte $s[B]$ und $t[B]$ der Attributmenge B in diesen Ausprägungen überein. $R(\mathcal{R})$ bezeichnet alle möglichen Relationen R des Relationenschemas \mathcal{R} . Folgende Tabelle verdeutlicht den Sachverhalt:

		R		
		A	B	C
r		a_1	b_2	c_1
s		a_2	b_1	c_2
t		a_2	b_1	c_1

Zu beachten ist, dass A, B und C Attributmengen (Keine einzelnen Attribute!) und a_i, b_i, c_i ($i \in \{1, 2\}$) entsprechende Wertetupel sind. Die Funktionale Abhängigkeit mehrerer Attribute lässt sich ebenfalls in der Mengenschreibweise darstellen: {Kundennr.} → {Name, Adresse}.

Als *triviale* funktionale Abhängigkeiten $A \rightarrow B$ bezeichnet man solche, bei denen die Attributmenge B Teilmenge der Attributmenge A ist. Jede Attributmenge bestimmt sich also selbst funktional (trivial). Außerdem bestimmt $\{\text{Name, Adresse}\}$ natürlich $\{\text{Name}\}$. Das ist ebenfalls eine triviale funktionale Abhängigkeit.

Eine funktionale Abhängigkeit bezeichnet man als *voll*, wenn sie nicht verkleinert werden kann, das heißt wenn kein Attribut aus A entfernt werden kann und $A \rightarrow B$ gelten soll. Formal: $A \twoheadrightarrow B \Leftrightarrow A \rightarrow B$ und $\forall a_i \in A : A \setminus \{a_i\} \not\rightarrow B$. Wenn A alle Attribute eines Relationenschemas \mathcal{R} voll bestimmt ($A \twoheadrightarrow \mathcal{R}$), dann ist A ein *Schlüsselkandidat*. Jeder Schlüsselkandidat erfüllt die Eigenschaften für einen Primärschlüssel eines Relationenschemas (Seite 6).

Man könnte jetzt auf die Idee kommen sich die Ausprägungen eines Relationenschemas einer Datenbank herzunehmen und die einzelnen Attribute zu untersuchen, ob sich diese nicht möglicherweise funktional bestimmen. Aber das wäre Unsinn! Warum?

Funktionale Abhängigkeiten ergeben sich nicht aus einer gegebenen Relation, sondern aus der Logik der Attribute eines Relationenschemas. Erkenntnisse über Funktionale Abhängigkeiten gewinnt man also nicht indem man die Werte der Attribute begutachtet, sondern indem man die Attribute in einen logischen Zusammenhang einordnet.

14 Armstrong-Axiome und kanonische Überdeckung

Bleiben wir beim Beispiel des Versandhandels:

Versand(Kundennr., Auftragsnr., Artikelnr., Name, Adresse, Ware, Preis)

Es enthält einige offensichtliche funktionale Abhängigkeiten:

1. $\{\text{Kundennr., Auftragsnr., Artikelnr.}\} \rightarrow \{\text{Name, Adresse, Ware, Preis}\}$
2. $\{\text{Kundennr.}\} \rightarrow \{\text{Name, Adresse}\}$
3. $\{\text{Auftragsnr.}\} \rightarrow \{\text{Kundennr.}\}$
4. $\{\text{Artikelnr.}\} \rightarrow \{\text{Ware}\}$
5. $\{\text{Auftragsnr., Artikelnr.}\} \rightarrow \{\text{Preis}\}$

Aus diesen funktionalen Abhängigkeiten — bezeichnen wir sie mit F — lassen sich weitere Abhängigkeiten ableiten. Die Menge aller funktionalen Abhängigkeiten die aus F ableitbar sind heißt Hülle von F und wird mit F^+ bezeichnet. Zur Berechnung dieser Hülle kann man die *Armstrong-Axiome* verwenden:

- Reflexivität: Sei $B \in A$, dann $A \rightarrow B$ (triviale funktionale Abhängigkeiten) und damit gilt $A \rightarrow A$.
- Verstärkung: Wenn $A \rightarrow B$, dann gilt auch $A \cup C \rightarrow B \cup C$
- Transitivität: Wenn $A \rightarrow B$ und $B \rightarrow C$, dann gilt auch $A \rightarrow C$

Die folgenden Regeln sind dabei ebenfalls hilfreich:

- Vereinigung: Falls $A \rightarrow B$ und $A \rightarrow C$ gilt, dann gilt auch $A \rightarrow B \cup C$
- Dekomposition: Falls $A \rightarrow B \cup C$ gilt, dann gilt auch $A \rightarrow B$ und $A \rightarrow C$
- Pseudotransitivität: Falls $A \rightarrow B$ und $B \cup C \rightarrow D$ gilt, dann gilt auch $A \cup C \rightarrow D$

Einige Beispiele:

- {Artikelnr., Preis} \rightarrow {Ware, Preis} (Verstärkung)
- {Auftragsnr.} \rightarrow {Name, Adresse} (Transitivität)
- {Auftragsnr.} \rightarrow {Kundennr., Name, Adresse} (Vereinigung)
- {Kundennr.} \rightarrow {Name} (Dekomposition)

Auf diese Weise kann man mehr Abhängigkeiten aufstellen, als einem möglicherweise lieb ist, denn beim Einfügen eines neuen Tupels in die Datenbank müssen alle diese Bedingungen überprüft werden. Und deshalb stampft man die ganzen funktionalen Abhängigkeiten am besten wieder ein. Doch halt! Um trotzdem die nötigen Bedingungen an die Datenbank zu stellen wäre es vielleicht sinnvoll sich nach einer kleinsten möglichen Menge funktionaler Abhängigkeiten umzusehen, die genau das selbe leistet. Und genau das tun wir!

Zwei Mengen F und G von funktionalen Abhängigkeiten eines Relationenschemas sind genau dann äquivalent, wenn sie die gleiche Hülle haben: $F \equiv G \Leftrightarrow F^+ = G^+$.

Diese möglichst kleine Menge funktionaler Abhängigkeiten zu einer gegebenen Menge F nennt man *kanonische Überdeckung* F_c . Sie muss die folgenden Eigenschaften erfüllen:

- $F_c \equiv F$, also $F_c^+ = F^+$
- Für alle funktionalen Abhängigkeiten $A \rightarrow B$ in F_c gibt es keine überflüssigen Attribute in A und in B . Das heißt:
 - $\forall a_i \in A : (F_c \setminus \{A \rightarrow B\} \cup \{(A \setminus \{a_i\}) \rightarrow B\}) \neq F_c$
 - $\forall b_i \in B : (F_c \setminus \{A \rightarrow B\} \cup \{A \rightarrow (B \setminus \{b_i\})\}) \neq F_c$
- Jede linke Seite der funktionalen Abhängigkeiten kommt in F_c nur einmal vor: Falls $A \rightarrow B$ und $A \rightarrow C$, dann wird in F_c nur $A \rightarrow B \cup C$ verwendet (Vereinigungsregel).

Um zu einer gegebenen Menge funktionaler Abhängigkeiten F die kanonische Überdeckung F_c zu finden, geht man so vor:

1. Linksreduktion: Für jede funktionale Abhängigkeit $A \rightarrow B$ in F in der ein Attribut $a_i \in A$ überflüssig ist ($B \subset \text{Hülle}(F, A \setminus \{a_i\})$) ersetze $A \rightarrow B$ durch $A \setminus \{a_i\} \rightarrow B$.
2. Rechtsreduktion: Für alle verbliebenen funktionalen Abhängigkeiten $A \rightarrow B$ in F in welchen ein Attribut $b_i \in B$ überflüssig ist ($b_i \in \text{Hülle}(F \setminus \{A \rightarrow B\} \cup \{A \rightarrow B \setminus \{b_i\}, A)$) ersetze $A \rightarrow B$ durch $A \rightarrow B \setminus \{b_i\}$.
3. Entferne die funktionalen Abhängigkeiten der Form $A \rightarrow \emptyset$.
4. Ersetze alle funktionalen Abhängigkeiten der Form $A \rightarrow B_1, A \rightarrow B_2, \dots, A \rightarrow B_n$ durch $A \rightarrow B_1 \cup B_2 \cup \dots \cup B_n$.

Um so viele Formalitäten zu verdauen sollten wir am besten ein Exempel statuieren. Gegeben seien dieses Relationenschema und die folgenden funktionalen Abhängigkeiten:

$$R(A, B, C, D, E, F)$$

$$F = \{A \rightarrow C \cup D, \quad B \rightarrow D \cup E \cup F, \quad C \cup D \rightarrow A \cup B \cup E, \quad D \rightarrow C \cup F, \quad F \rightarrow D \cup E\}$$

1. Die Linksreduktion kann wenn überhaupt nur dort ausgeführt werden, wo links mindestens zwei Attribute stehen. Das ist lediglich der Fall bei $C \cup D \rightarrow A \cup B \cup E$. Es gilt außerdem D bestimmt C wegen $D \rightarrow C \cup F$. Deshalb ist C in der ersten Regel überflüssig und kann entfernt werden.

$$F \equiv \{A \rightarrow C \cup D, \quad B \rightarrow D \cup E \cup F, \quad D \rightarrow A \cup B \cup E, \quad D \rightarrow C \cup F, \quad F \rightarrow D \cup E\}$$

2. Für jede der Abhängigkeiten wird nun die Rechtsreduktion ausgeführt.

- $F \rightarrow D \cup E$: Aus $D \rightarrow A \cup B \cup E$ geht hervor: D bestimmt E funktional. Deshalb verkürzt sich diese Regel zu $F \rightarrow D$.
 $F \equiv \{A \rightarrow C \cup D, B \rightarrow D \cup E \cup F, D \rightarrow A \cup B \cup E, D \rightarrow C \cup F, F \rightarrow D\}$
- $D \rightarrow C \cup F$: Aus den ersten drei Abhängigkeiten geht bereits hervor: (1) D bestimmt B und das wiederum bestimmt F und (2) D bestimmt A und das wiederum bestimmt C . Deshalb verkürzt sich diese Regel zu $D \rightarrow \emptyset$.
 $F \equiv \{A \rightarrow C \cup D, B \rightarrow D \cup E \cup F, D \rightarrow A \cup B \cup E, D \rightarrow \emptyset, F \rightarrow D\}$
- $D \rightarrow A \cup B \cup E$: Aus $B \rightarrow D \cup E \cup F$ ergibt sich: B bestimmt E . Damit verkürzt sich diese Abhängigkeit zu $D \rightarrow A \cup B$.
 $F \equiv \{A \rightarrow C \cup D, B \rightarrow D \cup E \cup F, D \rightarrow A \cup B, D \rightarrow \emptyset, F \rightarrow D\}$
- $B \rightarrow D \cup E \cup F$: Aus $F \rightarrow D$ geht hervor: F bestimmt D funktional. Deshalb verkürzt sich diese Regel zu $B \rightarrow E \cup F$.
 $F \equiv \{A \rightarrow C \cup D, B \rightarrow E \cup F, D \rightarrow A \cup B, D \rightarrow \emptyset, F \rightarrow D\}$
- $A \rightarrow C \cup D$: Diese Abhängigkeit kann nicht weiter verkürzt werden.

3. Im dritten Schritt werden noch die Regeln der Form $A \rightarrow \emptyset$ entfernt.

$$F \equiv \{A \rightarrow C \cup D, B \rightarrow E \cup F, D \rightarrow A \cup B, F \rightarrow D\}$$

4. Da es keine Abhängigkeiten mehr gibt auf deren linker Seite dieselben Attribute stehen, kann nicht weiter minimiert werden. $F_c = \{A \rightarrow C \cup D, B \rightarrow E \cup F, D \rightarrow A \cup B, F \rightarrow D\}$

15 Anomalien

Wie im Beispiel des Versandhandels bereits angesprochen, kann es in Datenbanken mit schlecht entworfenen Relationenschemata schnell zu Anomalien kommen. Hier noch einmal das ursprüngliche Schema:

Versand(Kundennr., Auftragsnr., Artikelnr., Name, Adresse, Ware, Preis)

Es gibt drei Arten von Anomalien:

Updateanomalien: Wenn sich die Adresse eines Kunden ändert, dann muss diese Änderung in jedem Tupel in dem dieser Kunde vorkommt, ausgeführt werden. Dabei besteht die Gefahr, dass Tupel übersehen werden und die Datenbank inkonsistent wird. Außerdem führt dies zu einem erhöhten Speicherbedarf und Leistungseinbußen bei Updates.

Einfügeanomalien: Wenn man eine neue Ware mit Artikelnummer in die Datenbank einfügen will, die noch nicht von einem Kunden gekauft wurde müssen die verbleibenden Attribute mit Nullwerten gefüllt werden. Das würde in diesem Fall sogar die Bedingungen für den Primärschlüssel verletzen und wäre somit unmöglich. Es ist also zu beachten, dass man in einem Relationenschema nicht mehrere Entitytypen aus der realen Welt vermischt.

Löschanomalien: Wenn man einen Kunden löschen möchte, kann es dann zu einem Problem kommen, wenn dieser Kunde der einzige war, der eine bestimmte Ware gekauft hat. Würde man dann den gesamten Eintrag des Kunden löschen, dann wären auch die Informationen zu dieser Ware verloren. Andererseits

könnte man die Attribute Kundennummer, Name und Adresse auch einfach mit Nullwerten überschreiben. Dann würde die Datenbank aber schnell an Größe zunehmen, da man keine Einträge löschen kann.

Um diese Anomalien zu vermeiden kann ein Relationenschema in eine *Normalform* gebracht werden. Dazu wird ein Schema \mathcal{R} in mehrere Relationenschemata $\mathcal{R}_1, \dots, \mathcal{R}_n$ zerlegt. (Genau wie die Zerlegung von Versand in Kunde und Auftrag am Anfang des vorigen Kapitels.) Dabei gelten zwei Anforderungen:

- **Verlustlose Zerlegung:** Ein Informationsverlust muss vermieden werden, das heißt die Relation $R(\mathcal{R})$ muss sich aus den Relationen $R(\mathcal{R}_1), \dots, R(\mathcal{R}_n)$ rekonstruieren lassen. Wenn ein Relationenschema \mathcal{R} in \mathcal{R}_1 und \mathcal{R}_2 zerlegt wird, so muss also gelten: $R = \pi_{\mathcal{R}_1}(R) \bowtie \pi_{\mathcal{R}_2}(R)$. Die Zerlegung von Versand in Kunde und Auftrag ist verlustlos.
- **Abhängigkeitsbewahrung:** Die funktionalen Abhängigkeiten müssen auch für das zerlegte Relationenschema gelten. Damit zur Überprüfung der Abhängigkeiten aus Effizienzgründen nicht jedesmal der natürliche Join über alle Relationenschemata ausgeführt werden muss, sollten alle funktionalen Abhängigkeiten lokal auf den zerlegten Schemata überprüfbar sein.

16 Normalformen

Wie bereits angesprochen ist das Ziel der Normalformen eine formale Definition für ein „gutes“ Datenbankdesign.

Erste Normalform: Ein Relationenschema ist in erster Normalform, wenn alle Attribute nur atomare Werte enthalten können. Es sind zum Beispiel keine mengenwertigen Attribute und keine geschachtelten Relationen möglich. Nach der bisher verwendeten Definition sind also bereits alle Relationenschemata in erster Normalform. Um dies zu verdeutlichen hier zwei Beispiele für Datenbanken, die nicht in erster Normalform sind:

mengenwertige Attribute			geschachtelte Relationen			
Vater	Mutter	Kinder	Vater	Mutter	Kinder	
Johann	Martha	{Else, Lucia}	Johann	Martha	KName	KAlter
Johann	Maria	{Theo, Josef}	Johann	Maria	Else	5
Heinz	Martha	{Cleo}	Johann	Maria	Lucia	3
			Johann	Maria	Theo	3
			Heinz	Martha	Josef	1
			Heinz	Martha	Cleo	9

Die Beispiele sind aus dem Buch „Datenbanksysteme - Eine Einführung“ von Alfons Kemper und André Eickler¹, welches ich sehr empfehlen kann. Wie man diese Relationen in erste Normalform bringt ist glaube ich trivial, oder nicht?

Zweite Normalform: Ein Relationenschema ist dann in zweiter Normalform, wenn mindestens *eine* dieser Bedingungen gilt:

- Jedes Attribut ist Teil eines Schlüsselkandidaten.
- Jedes Attribut ist von jedem Schlüsselkandidaten voll funktional abhängig.

¹Wenn man mir Geld dafür geboten hätte, dass ich das Buch hier empfehle, dann hätte ich auch die ISBN noch angegeben.

- Alle Nicht-Schlüsselattribute hängen vom ganzen Primärschlüssel ab.

Das Beispiel des Versandhandels könnte man so in zweiter Normalform darstellen:

Kunde(Kundennr., Name, Adresse)
 Bestellung(Auftragsnr., Kundennr.)
 Auftrag(Auftragsnr., Artikelnr., Preis)
 Artikel(Artikelnr., Ware)

Die Zerlegung ist natürlich verlustlos.

Dritte Normalform: Die dritte Normalform beseitigt Abhängigkeiten von Nicht-Schlüsselattributen. Ein Relationenschema \mathcal{R} ist in der dritten Normalform, wenn für alle Abhängigkeiten $A \rightarrow a_i$ mit $A \subset \mathcal{R}$, $a_i \in \mathcal{R}$, $a_i \notin A$ gilt:

- A enthält einen Schlüssel von \mathcal{R} oder
- a_i ist Teil eines Schlüsselkandidaten.

Um ein Relationenschema in dritte Normalform zu zerlegen gibt es einen *Synthesealgorithmus*. Dieser erfüllt die Bedingungen, dass kein Informationsverlust auftritt, dass die funktionalen Abhängigkeiten bewahrt werden und natürlich dass alle Relationenschemata die dritte Normalform erfüllen.

Gegeben seien ein Relationenschema \mathcal{R} und eine Menge funktionaler Abhängigkeiten F .

1. Bestimme die kanonische Überdeckung F_c zu F , also
 - (a) Linksreduktion der formalen Abhängigkeiten
 - (b) Rechtsreduktion der formalen Abhängigkeiten
 - (c) Entfernung von formalen Abhängigkeiten der Form $A \rightarrow \emptyset$
 - (d) Zusammenfassen von formalen Abhängigkeiten mit gleichen linken Seiten
2. $\forall A \rightarrow B \in F_c$ erzeuge ein Relationenschema \mathcal{R}_A und ordne diesem die funktionalen Abhängigkeiten $F_A = \{C \rightarrow D \in F_c \mid C \cup D \subseteq \mathcal{R}_A\}$ zu.
3. Falls alle der in Schritt 2 erzeugten Schemata keinen Schlüsselkandidaten des ursprünglichen Schemas \mathcal{R} enthalten, so erzeuge zusätzlich eine Relation mit dem Schema $\mathcal{R}_K = K$ und $F_K = \emptyset$, wobei K ein Schlüsselkandidat von \mathcal{R} ist.
4. Eliminiere die Schemata \mathcal{R} , die in einem anderen Schema enthalten sind.

Und das gilt es nun am Beispiel des Versandhandels durchzuexerzieren. Gegeben ist also:

Versand(Kundennr., Auftragsnr., Artikelnr., Name, Adresse, Ware, Preis) und
 $F = (f_1, f_2, f_3, f_4, f_5)$ mit
 $f_1 = \{\text{Kundennr.}, \text{Auftragsnr.}, \text{Artikelnr.}\} \rightarrow \{\text{Name}, \text{Adresse}, \text{Ware}, \text{Preis}\}$
 $f_2 = \{\text{Kundennr.}\} \rightarrow \{\text{Name}, \text{Adresse}\}$
 $f_3 = \{\text{Auftragsnr.}\} \rightarrow \{\text{Kundennr.}\}$
 $f_4 = \{\text{Artikelnr.}\} \rightarrow \{\text{Ware}\}$
 $f_5 = \{\text{Auftragsnr.}, \text{Artikelnr.}\} \rightarrow \{\text{Preis}\}$

1. Kanonische Überdeckung
 - (a) Linksreduktion:
 $f_1 = \{\text{Auftragsnr.}, \text{Artikelnr.}\} \rightarrow \{\text{Name}, \text{Adresse}, \text{Ware}, \text{Preis}\}$

(b) Rechtsreduktion:

$$f_1 = \{\text{Auftragsnr.}, \text{Artikelnr.}\} \rightarrow \{\text{Preis}\}$$

$$f_5 = \{\text{Auftragsnr.}, \text{Artikelnr.}\} \rightarrow \emptyset$$

(c) f_5 kann entfernt werden

(d) Es können keine formalen Abhängigkeiten zusammengefasst werden.

2. Relationenschemata erzeugen:

$\mathcal{R}_1(\text{Auftragsnr.}, \text{Artikelnr.}, \text{Preis})$

$\mathcal{R}_2(\text{Kundennr.}, \text{Name}, \text{Adresse})$

$\mathcal{R}_3(\text{Auftragsnr.}, \text{Kundennr.})$

$\mathcal{R}_4(\text{Artikelnr.}, \text{Ware})$

3. Die Relation \mathcal{R}_1 enthält den Primärschlüssel (und damit einen Kandidatenschlüssel) des ursprünglichen Schemas.

4. Es kommt kein Relationenschema in einem anderen vor, also muss auch keines eliminiert werden.

Wenn ich nichts übersehen habe, dann ist das Schema in Schritt 2 in dritter Normalform.

Schlusswort

Soweit so gut. Ich hoffe, dass das ein Anfang ist, um erfolgreich zu lernen. Ich möchte nochmal dazu Auffordern mir Fehler zu melden, denn möglicherweise liest das auch einmal jemand, der den / die Fehler nicht erkennt und dann etwas falsches lernt.

Viel Erfolg bei der Prüfung!
