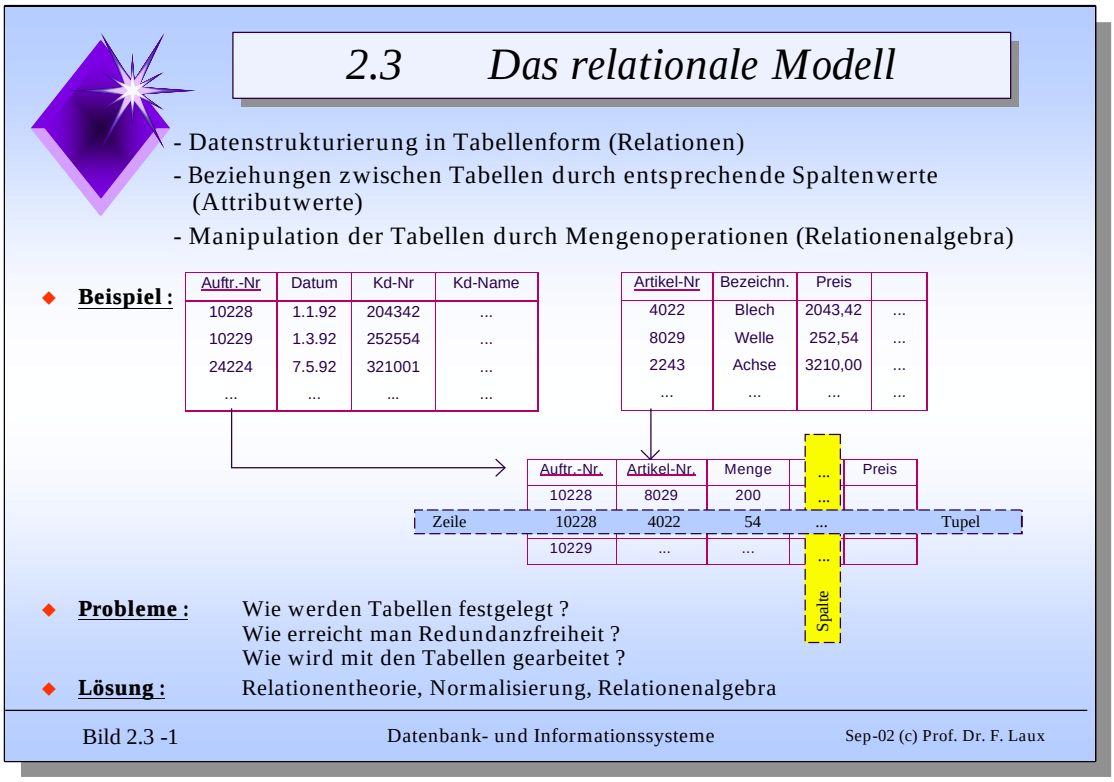


Die mit Abstand bekannteste Datenstrukturierung ist das weitverbreitete **relationale Modell**. Es führt alle Datenstrukturen des zu modellierenden Sachverhalts auf *Relationen* zurück. Dabei werden die Eigenschaften eines Objekts als Attribute von Relationen abgebildet. Normalisierte Relationen können anschaulich durch zweidimensionale *Tabellen* dargestellt werden. Die Beliebtheit dieses Modell für den kommerziellen Einsatz erklärt sich nicht nur aus der Tatsache, dass viele betriebliche Informationen ebenfalls tabellarisch dargestellt werden, sondern auch aus der formalen Schlichtheit des Modells und seiner Flexibilität.

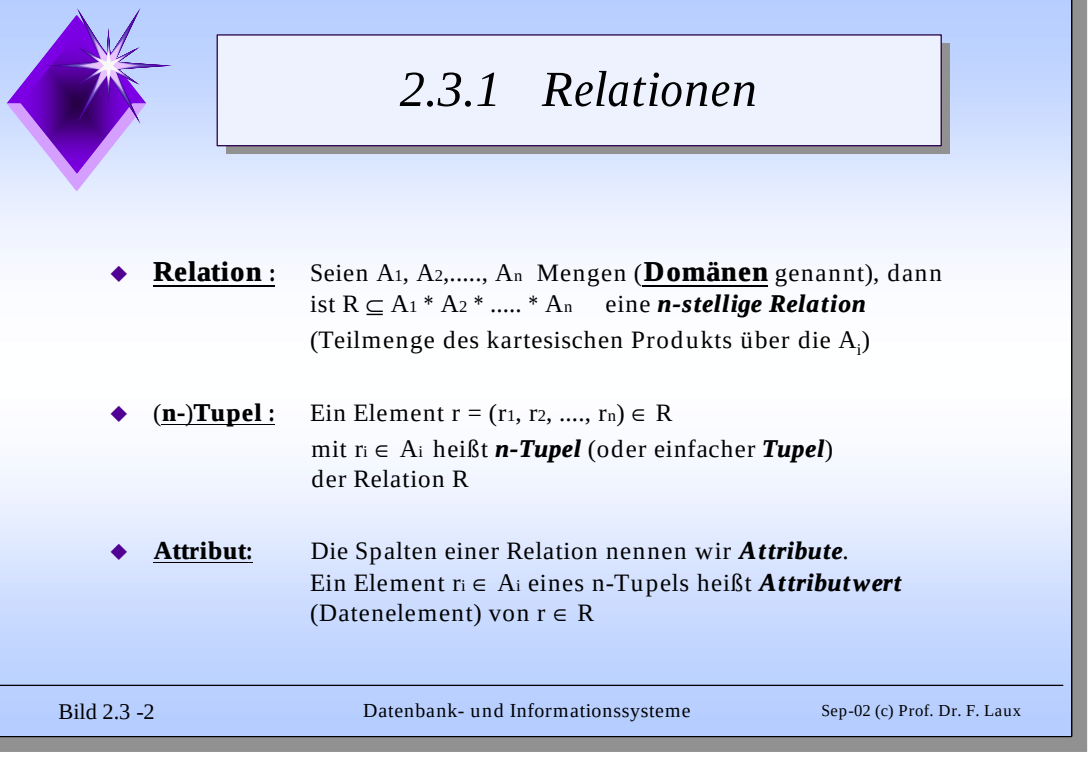
Wenn wir das Beispiel von 2.3-1 betrachten, fällt auf, dass sich unser relationales Modell auf 2 betriebliche Begriffe (Objekte) bezieht: Artikel und Auftag. Die Artikelrelation ist eine direkte Abbildung der betrieblichen Situation, während ein Auftrag durch 2 Relationen dargestellt wird. Wir können an diesem einfachen Beispiel schon erkennen, dass das Modell vor allem für Sachverhalte geeignet ist, die sich in natürlicher Weise tabellarisch darstellen lassen.

Die *Beziehungen* der Relationen zueinander werden über korrespondierende Attributwerte definiert. Diese Strukturinformationen müssen bei jeder Abfrage neu aufgebaut werden, was zu Kritik bezüglich der Effizienz von Datenrecherchen geführt hat. Soll z. B. ein Auftrag rekonstruiert werden, so müssen in unserem Beispiel Daten aus drei Relationen (wieder) zusammengefügt werden. Dies geschieht durch Mengenoperationen auf den Relationen (*Relationenalgebra, Bild* 2.3-7).

Wir nennen ein Element einer Relation *Tupel*. Jede *Zeile* einer Tabelle entspricht einem Tupel der Relation. Die möglichen Werte eines Attributs nennen wir seinen Wertebereich. In einer tabellarischen Darstellung entstammen alle Attributwerte (Spalte) dem Wertebereich des Attributs.

Komplexe Information kann auf verschiedene Arten strukturiert werden. Nur wenige Strukturen sind "gut". Die Relationentheorie liefert uns die Regeln dafür.

ÜA: In unserem Beispiel hätte der Auftrag auch durch eine Relation dargestellt werden können. Wie müsste die Relation dann aussehen? Welche Nachteile sehen Sie dabei?



Zur Präzisierung des Begriffs der Relation legen wir folgendes fest:

Eine *Domäne* (*Wertebereich*) Ai repräsentiert die Menge der zulässigen Werte eines Attributs. Die Werte können beliebige Daten darstellen.

Eine *Relation* R auf den Domänen A_1, A_2, ... A_n ist definiert als Teilmenge des kartesischen Produkts X A_i (i=1 ... n).

Ein Element r von R heißt *Tupel* mit den Komponenten $r_i \in A_i$.

Vereinfacht kann man eine Relation als *zweidimensionale Tabelle* auffassen. Dabei entsprechen die *Spalten* den Domänen, die *Zeilen* den Tupeln und ein Tabellenwert repräsentiert einen Attributwert. Wird jede Spalte mit einem Bezeichner, dem Attributnamen, versehen, so wird die Spaltenreihenfolge bedeutungslos. Dieser Vergleich ist allerdings nicht ganz exakt, denn die Tupel einer Relation bilden eine Menge, können demzufolge keine Dubletten aufweisen, während eine Tabelle gleiche Zeilen enthalten kann.

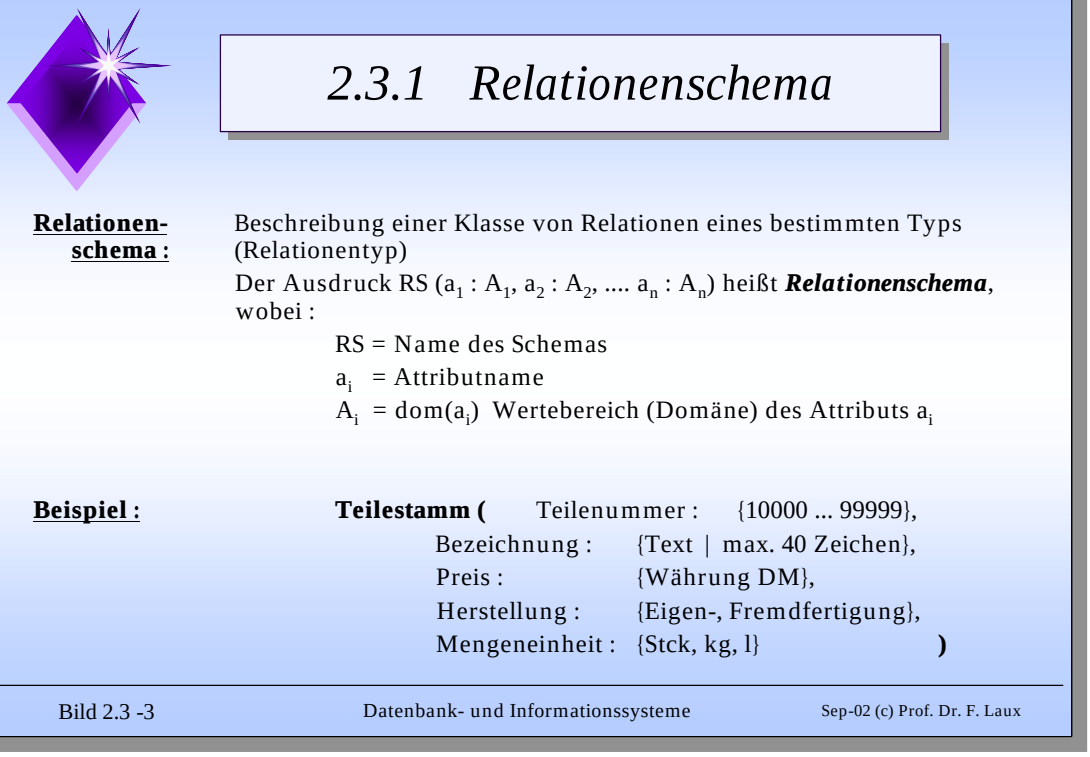
Beispiele: Domänen:
- N = Menge der natürlichen Zahlen
- Q = Menge der rationalen Zahlen
- string = Menge der Zeichenfolgen
- [0:120] = Alle Zahlen im Intervall 0 bis 120

Attribute: Name, Alter, Budget, Personal-Nr

Relation: Mitarbeiter (Personal-Nr, Name, Alter)

Personal-Nr	Name	Alter	
1234	Einstein	76	<- Tupel
0007	Bond	36	<- Tupel
0815	Nobody	22	<- Tupel

ÜA: Weisen Sie den Attributen *Name*, *Alter*, *Budget*, *Personal-Nr* Domänen zu.



Im Hinblick auf die Datenmodellierung ist eine zeitinvariante Beschreibung von Relationen wichtig, denn wir wollen nicht die augenblickliche Situation sondern ihre statische Struktur ausdrücken.

Relationenschema :

Wir definieren ein *Relationenschema* als Beschreibung (Definition) einer Klasse gleichartiger Relationen. Seine Definition enthält den *Namen des Schema*, die *Attributnamen* und ihre *Domänen*. D. h. eine Relation ist eine mögliche Ausprägung (ein Exemplar) eines Relationenschemas.

Attribut : Jede Menge A_i des RS wird eindeutig durch einen Namen identifiziert. Dieser Name wird als ***Attribut(name)*** des RS bezeichnet

Domäne : Jede Menge A_i des RS wird als ***Wertebereich (Domäne)*** des Attributs a_i bezeichnet.

Bei den Grundbegriffen zu Kap. 2 haben wir zwischen Exemplar und seiner Beschreibung bzw. Struktur unterschieden. Relation und Relationenschema sind ein weiteres Beispiel dafür:

Bsp: Relationenschema: *Mitarbeiter* (Personal-Nr:N, Name:string, Alter:[0:120]);
Attribute: Personal-Nr, Name, Alter;
zugehörige Domänen: N, string, [0:120]

Eine Ausprägung dieses Relationenschemas finden Sie auf der vorherigen Seite.

Wie wir bereits in einem Beispiel gesehen haben, wird die Realität durch mehrere, voneinander abhängigen Relationen beschrieben. Wir definieren deshalb ein
relationales Datenbankschema als ein Quadrupel **({RS}, {FD}, {DI}, DML)**, wobei
{RS} eine Menge von *Relationenschemata* ,
{FD} eine Menge von *funktionalen Abhängigkeiten* zwischen Attributen einer Relation,
{DI} eine Menge von *Abhängigkeiten zwischen Relationen (**referentielle Integrität**)* darstellt und
DML eine relationale *Manipulationssprache* bezeichnet.

Bsp: Datenbankschema: **Projektverwaltung** (vgl. ERM auf Bild 2.1-5)

RS: Schema 1: *Mitarbeiter* (Pers-Nr:N, Name:string)
Schema 2: *Projekt* (Proj-Name:string, Budget:DM)
Schema 3: *arbeitet* (MA-Nr:N, Proj-Name:string, Stunden:decimal)

(N = natürliche Zahlen, string = Zeichenfolgen, DM = Deutsche Mark (Währung), decimal = reelle Zahlen)

FD: Pers-Nr -> Name (es gibt eine Funktion f mit: Name = f(Pers-Nr))
Proj-Name -> Budget
MA-Nr, Proj-Name -> Stunden

DI: arbeitet.MA-Nr ∈ {Mitarbeiter.Pers-Nr} (Jeder Attributwert von MA-Nr in der Relation *arbeitet* muss auch in der Relation *Mitarbeiter* vorkommen)

Damit wird ausgesagt, daß nur Werte aus der Relationen *Mitarbeiter* zugelassen sind, d.h. der Wertebereich ist von Attributwerten einer anderen Relation abhängig und damit zeitlich veränderlich (*dynamisch*). Diesen Sachverhalt nennen wir **referentielle Integrität**.

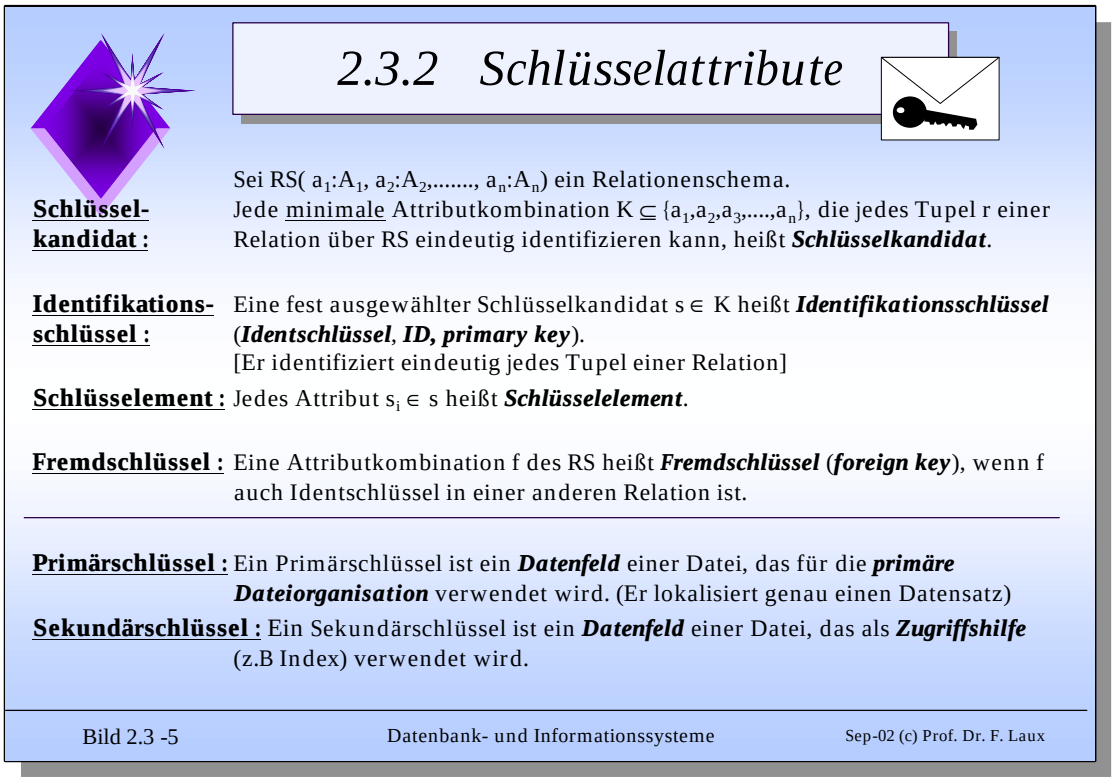
arbeitet.Proj-Name ∈ {Projekt.Proj-Name} (beachten Sie, dass die Namensgleichheit zwar nicht zufällig ist, aber formal keine Bedeutung (Semantik) besitzt, deshalb ist diese Bedingung ebenso notwendig wie die vorangehende.

Die referentiellen Bedingungen werden oft schon bei der Definition eines Relationenschemas angegeben. Z. B. kann die Relation *arbeitet* geschrieben werden als:

arbeitet (MA-Nr:Mitarbeiter.Pers-Nr, Proj-Name:Projekt.Proj-Name, Stunden:decimal)

DML: Relationenalgebra (mit den Mengenoperationen *Projektion, Vereinigung, Selektion, kartesisches Produkt*, etc., siehe Bild 2.3-7)

ÜA: Erstellen Sie von den Relationen in Bild 2.3-1 ein vollständiges Datenbankschema.



Zur Illustration der Schlüsselbegriffe betrachten wir unser Beispiel auf der vorherigen Seite, das relationale Datenbankschema ***Projektverwaltung***.

Das Attribut *Pers-Nr* ist der einzige Schlüsselkandidat der Relation *Mitarbeiter*, denn der Name ist nicht ausreichend, um ein Tupel zu identifizieren. Damit ist Pers-Nr auch der *Identifikationsschlüssel* (ID).

Der ID der Relation *arbeitet* ist die Attributmenge {MA-Nr, Proj-Name} (warum?). Beide Attribute werden auch als *Schlüsselelemente* bezeichnet, da sie zum ID gehören. Ausserdem sind sie auch *Fremdschlüssel* bezüglich den Relationen Mitarbeiter und Projekt.

Im Gegensatz zu den o.g. Schlüsseln sind *Primär- und Sekundärschlüssel* Begriffe der physischen Datenorganisation (Dateistruktur). In der Literatur wird dieser Unterschied gelegentlich verwechselt und der ID als Primärschlüssel bezeichnet. Dies hat vermutlich zwei Gründe:

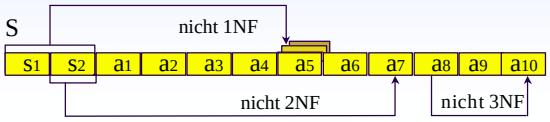
• in der englischen Literatur heißt der ID *primary key*

• der Primärschlüssel wird häufig auch als ID verwendet. (dies erleichtert die technische Realisierung der Eindeutigkeitsforderung für den ID)

Beispiel: Wenn die Matrikelnr zugleich die Satznummer in einer Studentendatei ist, wird automatisch sichergestellt, daß die Matrikelnr eindeutig ist, denn es gibt für eine Nummer nur eine physische Adresse. In unserem Beispiel hat dies den Nachteil, dass die Datei viele Lücken (nicht verwendete Matrikelnummern) aufweist. Aus diesem Gesichtspunkt wäre eine andere Organisation der Daten und damit eine Unterscheidung zwischen Primärschlüssel und ID günstig.

2.3.3 Normalformen

funktionale Abhängigkeit: Sei R (a1:A1, a2:A2,.... , an:An) ein RS und {aj} ⊆ {ai} , Ai = dom(ai) (i,j := 1,2, ... n)
Ein Attribut ak ∈ {ai} ist von der Attributmenge {aj} ***funktional abhängig***, wenn es eine skalare (einwertige) Funktion gibt mit: y = f(x) für alle y ∈ Ak und geeignetem x ∈ Xaj
Notation : {aj} -> ak



Sei S (z. Bsp. S = {s1,s2}) der Identschlüssel von RS.

1NF : Eine Relation R heißt ***einfach***, bzw. in ***erster Normalform***, wenn für alle Attribute ai gilt : S -> ai

2NF : Eine Relation R ist in **zweiter Normalform,** wenn sie in 1NF ist und keine funktionale Abhängigkeit von einer Teilmenge von S besteht. D.h. alle ai sind ***voll funktional*** von S (Identschlüssel) abhängig.

3 NF : Eine Relation R ist in **dritter Normalform**, wenn sie in 2NF ist und keine funktionale Abhängigkeit von Nichtschlüsselattributen an (∈ {aj} \ S) vorliegt.
Das bedeutet, es gibt ***keine transitiven Abhängigkeiten*** [S -> an -> ai] vom Identschlüssel.

Bild 2.3 -6 Datenbank- und Informationssysteme Sep-02 (c) Prof. Dr. F. Laux

Das Ziel der Normalisierung ist:

- Redundanzen zu verringern
- Anomalien zu vermeiden.

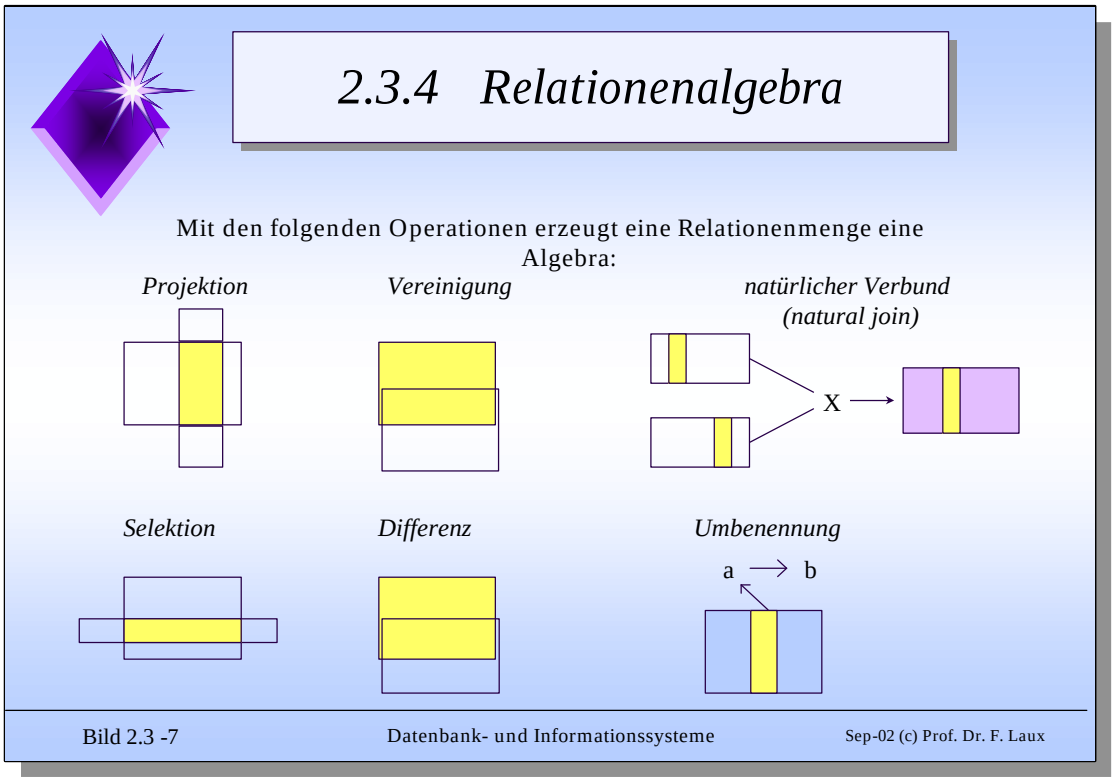
Betrachen wir das folgende Beispiel:

Relationenschema: *Auftrag* (Auftr.-Nr, Datum, Kd-Nr, Kd-Name, Kd-Adresse) (wir lassen die Domänen der Einfachheit halber weg und unterstreichen den ID)

Wenn ein Kunde mehrere Aufträge erteilt, muss jedesmal seine Nummer, Name und Adresse erfasst werden.

Diese Redundanz kann zu inkonsistenten Kundendaten führen: nehmen wir an, daß ein Kunde zwei Aufträge erteilt, dann müssen seine Daten 2 mal identisch erfasst werden. Ist dies nicht der Fall (z.B. können unterschiedliche Kundennummern oder Namen für einen Kunden erfasst werden) treten Widersprüche auf (z.B. ein Kunde wird als 2 verschiedene Kunden geführt oder ein Kunde ist unter verschiedenen Namen bekannt).

Wenn alle Aufträge eines Kunden gelöscht werden, verschwinden auch die Kundendaten (Löschanomalie). Es ist unmöglich, einen Kunden zu erfassen, ohne gleichzeitig auch einen Auftrag zu erzeugen.



Die *Relationenalgebra* (RA) dient als Maßstab für *Datenmanipulationssprachen* (DML). Obwohl sie keine kommerzielle Bedeutung besitzt, werden alle realisierten relationalen Sprachen mit dieser Algebra verglichen. Die RA bietet mengenbasierte Grundoperationen auf Relationen: Vereinigung, Durchschnitt (Selektion), Projektion, Differenz und Verbundoperation.

Der *natürliche Verbund* (*natural join*) ist eine Teilmenge des kartesischen Produktes zweier Relationen wobei Tupel mit korrespondierenden Attributen verknüpft werden.

Beispiel für einen natürlichen Verbund:

Seien folgende Relationen gegeben:

Auftrag :

(**AuftrNr**,	**Datum**,	**Artikel**,	**Kd_Nr**)
1	12/10/95	Welle	101
2	14/10/95	Schraube	101
3	16/10/95	Stahl ST37	102

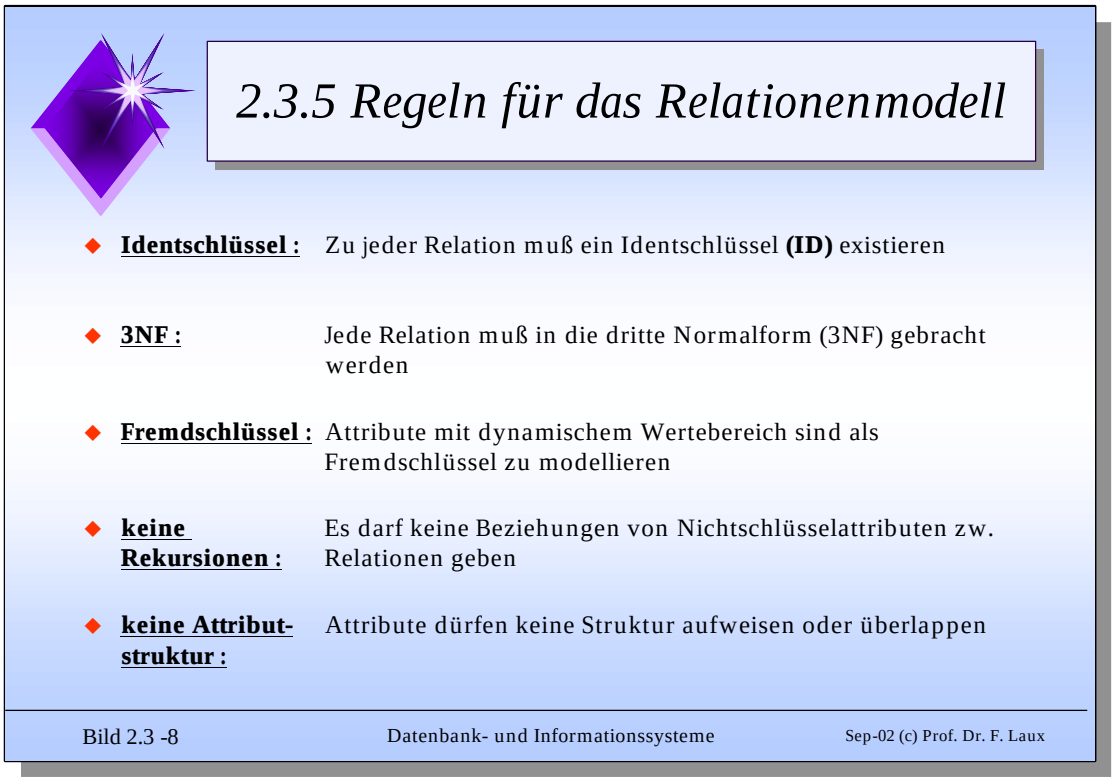
und

Kunde :

(**Kd_Nr**,	**Name**)
101	Mitterrand
102	Chirac

Der ***natürliche Verbund*** über den Fremdschlüssel Kd_Nr lautet dann:

(**AuftrNr** ,	**Datum**,	**Artikel**,	**Kd_Nr**	**Name**)
1	12/10/95	Welle	101	Mitterrand
2	14/10/95	Schraube	101	Mitterrand
3	16/10/95	Stahl ST37	102	Chirac

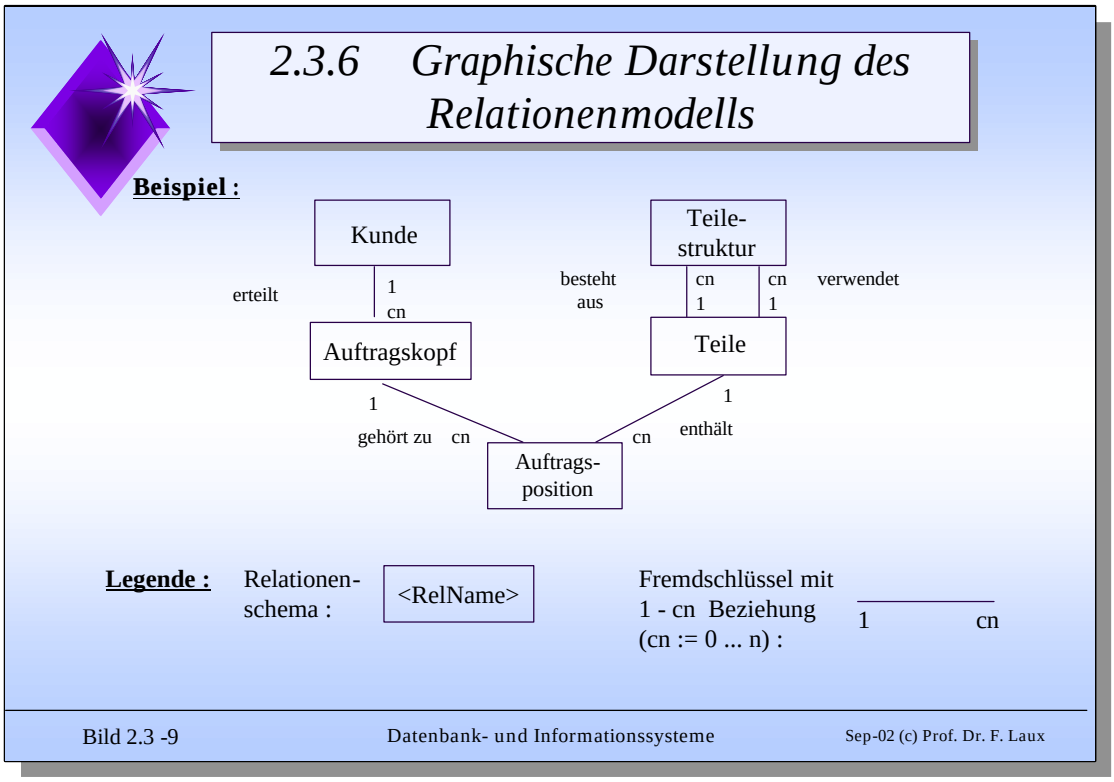


Die Datenmodellierung wird als eine intuitive, kreative Aufgabe angesehen, welche viel Erfahrung erfordert.

Trotzdem gibt es einige Regeln, die uns beim Modellierungsprozess unterstützen:

Der Ident-Schlüssel und andere funktionale Abhängigkeiten (FD) helfen uns bei der Normalisierung. Besonders zu beachten sind transitive Abhängigkeiten, welche uns zur 3. Normalform (3NF) und zu den Fremdschlüsseln führen. Im Sinne einer klaren, stabilen und widerspruchsfreien Datenstruktur sind Rekursionen von Nichtschlüsselelementen zu vermeiden.

Wenn ein Attribut eine Unterstruktur aufweist, sollte es in seine atomaren Komponenten zerlegt werden, denn sonst bleibt die Semantik dieser Struktur dem Modell verborgen.



Eine graphische Darstellung hilft uns, die Datenstrukturen leichter zu erfassen und gibt uns einen Überblick über das Datenmodell.

Wir verwenden das **Bachman Diagramm**, um Relationen und ihre Beziehungen graphisch darzustellen:
Ein Rechteck steht für ein Relationenschema und eine Verbindungslinie zwischen zwei Schemata zeigt eine Fremdschlüsselbeziehung an. Wir verwenden Pfeile oder Kardinalitätssymbole *1 - cn* um die Mächtigkeit (Anzahl Verbindungen der Tupel untereinander) der Beziehung anzuzeigen.

Unser Beispiel zeigt die Struktur eines Auftragsverwaltungssystems.

Ein Kunde kann mehrere Aufträge erteilen. Jeder Auftrag enthält eine oder mehrere Positionen, wobei jede Position genau einen Artikel umfasst. Artikel - außer Einzelteile - besitzen eine baumartige (hierarchische) Komponentenstruktur. Umgekehrt kann ein Teil in anderen Artikeln enthalten sein (Verwendungsnachweis).

Wie man leicht sieht, wird eine viele-zu-viele (n:m) Beziehung zwischen Auftrag und Artikel durch zwei 1:cn Beziehungen (gehört zu, enthält) und eine Beziehungsrelation (Auftragsposition) hergestellt. Diese Hilfskonstruktion ist nötig, da wir im relationalen Modell nur 1:cn Beziehungen darstellen können.

ÜA: Welche Beziehungstypen kennen Sie? (Hinweis: siehe 1.4, 2.2.3)