2 Datenbanksysteme

- Im Folgenden werden wir einige grundlegende Eigenschaften von Datenbanksystemen kennen lernen
- Datenbanken sind Bestandteil vieler Anwendungssysteme; sie stellen die dort benötigten Daten bereit
- Aufbau des Abschnitts
  - Definition grundlegender Begriffe
  - Datenmodelle
  - Relationale Datenmodelle
  - Entwurf einer Datenbank

Datenbanken…was ist das denn nun wieder?

2.1 Grundlegende Begriffe

- Eine Datenbank ist eine Ansammlung verwandter Fakten, die Bedeutung haben und gespeichert werden können; dies sind die Daten
  - Die Daten sind zusammenhängend und haben anhaltend Bedeutung
  - Die Daten werden angelegt, gepflegt und in einer speziellen Struktur gehalten
  - Die Daten dienen bestimmten Anwendungen und besitzen so eine Gruppe von gezielten Benutzern
- Dies ist allerdings keine eindeutige Definition sondern lediglich eine charakterisierende Beschreibung von bestimmten Eigenschaften

Schemata und Instanzen

- Unter einem **Schema** (oder auch Meta Daten) einer Datenbank versteht man die Beschreibung der Struktur der dort zu speichernden Daten
- Unter den **Ausprägungen dieser Datenbank** versteht man die konkreten Datenbestände, die in der Datenbank gespeichert sind
- Die konkreten Datenbestände werden auch **Zustand einer Datenbank** genannt
- Somit führen offensichtlich Veränderungen dieser Datenbestände zu neuen Zuständen der Datenbank

Datenbank Management System

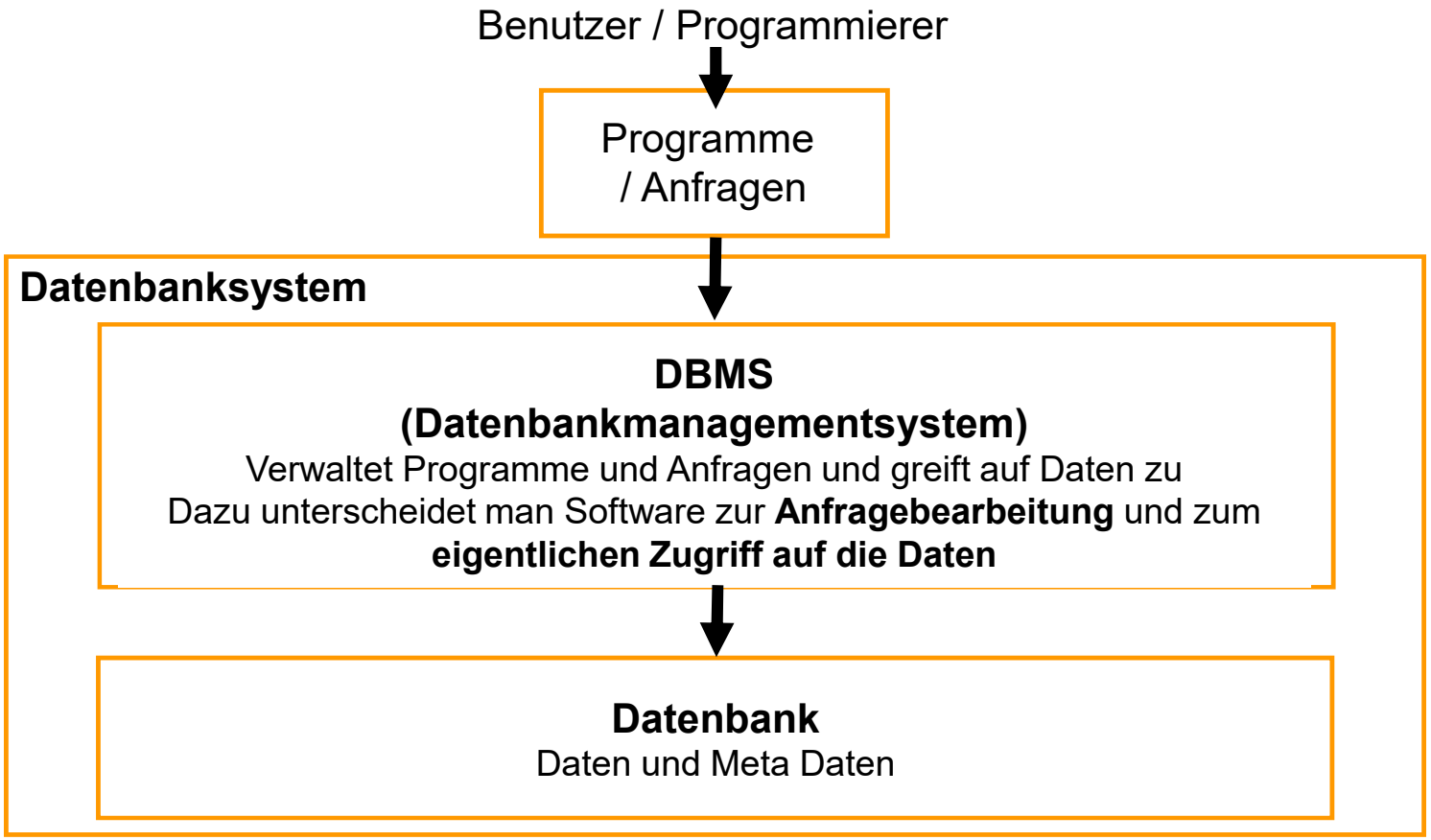
- Sammlung von Programmen, die es ermöglichen eine Datenbank zu erstellen und aufrecht zu erhalten
  - DB Sprache für Operationen auf Datenbeständen
  - Universell und anwendungsunabhängig
  - Sichert Konsistenz der Daten
  - Sichert Synchronisation
  - Bietet standardisierte Schnittstelle
- Dabei wird unterstützt
  - Definition der Datenbank
    - Festlegung der Datenstrukturen und
    - Bedingungen der Datenspeicherung
  - Manipulation der Datenbank
    - Auslesen und
    - Veränderung der Datenbestände
  - Konstruktion der Datenbank
    - Abspeicherung der Datenbestände
- Allgemein dienliche und einsetzbare Software
- Sie ist nicht erforderlich, um eine Datenbank zu verwalten, sondern man kann auch eigen entwickelte Software benutzen

Datenbank

- Die Datenbank besteht aus
  - Datenausprägungen (den eigentlichen Daten)
  - und Datenbeschreibungen (den Meta Daten, d.h. dem Datenbank Schema)
- Sie stellt den integrierten Datenbestand anhand eines einheitlichen Datenmodells geordnet dar
- Das heißt man unterscheidet
  - Die Datenbank Definition
  - und die dort gespeicherten eigentlichen Daten

Datenbanksystem

- Besteht aus einem Datenbankmanagementsystem und der Datenbank
- Schematisch



Warum gibt es Datenbanksysteme?

- Für die Beantwortung dieser wichtigen zentralen Frage wollen wir im Folgenden einen kurzen Einblick in die Entwicklungsgeschichte von Datenbanksystemen erarbeiten
- Hieraus wird schnell deutlich,
  - welche Vorteile Datenbanksysteme dem Nutzer bereitstellen
  - und welche Nachteile vorangegangener Systeme zu ihrer Entwicklung geführt haben

2.1.1 Entwicklungsgeschichte I

Stufe 1: Elementare Dateien

- Datenhaltung um 1960
- Anwendungen halten eigene Daten in Dateiform mit individueller Struktur
- Anwendungssysteme führen den Datenzugriff bis ins kleinste Detail selbst durch
- (Schwacher) Vorteil
  - Schnelle Speziallösungen sind möglich
- Nachteile
  - Inkonsistenz der Datenbestände kann durch unterschiedliche Dateien und ihrer Modifikation entstehen
  - Redundanz
  - Geräteabhängigkeit der Anwendungssysteme
  - „Extreme Expertenmacht“ kann entstehen

Entwicklungsgeschichte II

Stufe 2: Dateiverwaltungssysteme

- Datenhaltung um 1965
- Anwendungen halten auch hier eigene Daten in Dateien mit individueller Struktur
- Anwendungssysteme führen den Datenzugriff einheitlich mit Hilfe bestehender Betriebssystemroutinen durch
- Erreichter Vorteil
  - Geräteunabhängigkeit wird nun erreicht
- Nachteile
  - Abhängigkeit von den Speicherstrukturen
    - Betriebssystem liefert lediglich unspezifizierte Sätze zurück
    - Genaue Formatierung der Sätze muss das Anwendungssystem, d.h. der Anwender/Programmierer übernehmen
  - Inkonsistenz der Datenbestände kann durch unterschiedliche Dateien und ihrer Modifikation entstehen
  - Redundanz
  - Alle Update Operationen muss das Anwendungssystem in ihrem Funktionsumfang und zeitlichen Verhalten selbst individuell festlegen
  - Bei Strukturanpassungen sind diese Operationen mit zu ändern

Entwicklungsgeschichte III

Stufe 3: Datenbanksysteme

- Datenhaltung ab 1975
- Anwendungssysteme führen den Datenzugriff über Datenbanksysteme durch
- Dabei entsteht eine einheitliche Benutzungsschnittstelle zwischen Anwendung und dem Datenbankmanagementsystem
- Erreichter Vorteil
  - Keine oder nur gewollte Redundanz
  - Zentrale Kontrolle der Benutzung und Art der Veränderung
  - Anwendung arbeitet unabhängig von den konkreten Daten, d.h. es erfolgt kein direkter Zugriff mehr auf untere Schichten (direkt auf die Daten und ihre Strukturen)
- Nachteile
  - Durch zentrale Datenhaltung rückt das Thema „Datenschutz“ als neues Problem in den Mittelpunkt der Betrachtung
  - Durch zentrale Datenhaltung sind spezielle Probleme bei erforderlichen Zugriffsgeschwindigkeiten zu berücksichtigen

Leistungen von Datenbanksystemen

- Integration der Datenbestände

  Alle Daten werden zentral an einer Stelle einheitlich abgelegt und verwaltet
- Angebot an vordefinierten standardisierten Operationen zur Datenvisualisierung und -verarbeitung
- Beschreibung von Datenbeständen (Definition der Datenbestände sind ebenfalls veränderbar)
- Benutzersichten
  - Jeder Benutzer kann eine eigene Sicht auf Daten erhalten
  - D.h. er erhält als Mitglied einer Gruppe Zugriff auf bestimmte Teile der gesamten Datenbank
- Konsistenz und Integritätsüberwachung
  - Korrekte Datenbank Inhalte und Veränderungen werden ermöglicht
  - Keine Überprüfung durch Anwendungen notwendig

Leistungen von Datenbanksystemen

- Datenschutz durch die Verwaltung von Zugriffsrechten
- Transaktionsmanagement
  - Zusammenfassung von zusammengehörigen Aktionen zu einer unteilbaren Einheit
  - Ermöglicht eindeutige Ergebnisse von Update Operationen
- Synchronisation
  - Koordination von konkurrierenden Zugriffen
  - D.h. Verwaltung gleichzeitiger Transaktionen mehrerer Benutzer auf die vorhandenen Datenbestände
- Datensicherung
  - Laufende Datensicherung
  - Wiederherstellung der Datenbestände nach Systemfehlern
- Unabhängigkeit zwischen Daten und Programmen (Bei OO Datenbanken zudem Programm und Methoden)

2.1.2 Datenbankbenutzer

- **Datenbank Administrator (DBA)**
  - Verantwortlich für Entwurf, Überwachung und Anpassung eines Datenbanksystems
  - Damit ergeben sich die grundlegenden Aufgaben
    - Definition der konzeptionellen / externen / internen Schemata (siehe unten)
    - Datenschutz
    - Leistungsüberwachung (ggf. Änderungen des internen Schemas)
    - Überwachung der Anforderungen (ggf. Änderungen des externen Schemas)
    - Betreuung/Verwaltung von Benutzern
- **Datenbank Designer (DBD)**
  - Sammelt bei der Entwicklung der erforderlichen Schemata Informationen
  - Unterstützt den Datenbank Administrator

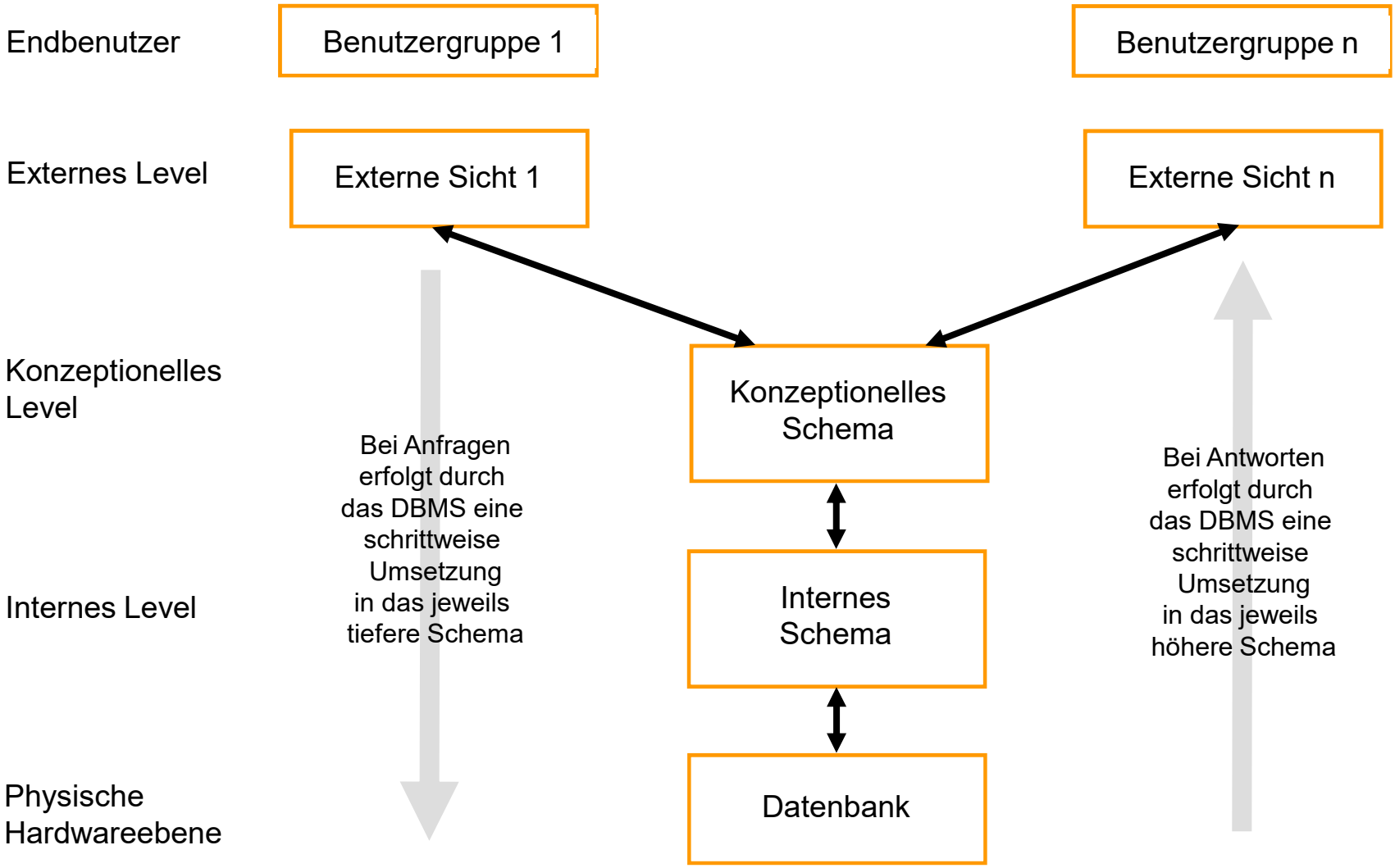
Datenbankbenutzer – Fortsetzung

- **System Analyst**
  - Analyse von Anforderungen der Anwendungsseite
  - Bereitet Spezifikation für Anwendungsprogramme vor
- **Anwendungsprogrammierer**
  - Entwickeln Anwendungsprogramme mit Datenbank Zugriffen
  - Nutzt Spezifikation der System Analysten
- **Hardware/Software Spezialisten**
  - Sorgen für lauffähige Gesamtsysteme
  - Haben aber keine direkten Datenbankaufgaben
- **End Benutzer**
  - Gehobene
    - Datenbank Anfragen auf der Basis prozeduraler Sprachen
    - Ggf. mit Sonderrechten
  - Einfache Benutzer
    - Tätigen meist Standardanfragen
    - Nutzen ggf. als valide eingestufte Standardprogramme

2.1.3 Drei Schemata Architektur

- Ziel: Strikte Trennung der Anforderungen und Anwendungen der Benutzer von der physikalischen Datenbank
- Daher finden sich Schemata auf insgesamt drei verschiedenen Ebenen
- Man unterscheidet
  - Das interne Level (mit dem internen Schema)
  - Das konzeptionelle Level (mit dem konzeptionellen Schema)
  - Das externe Level (mit den externen Schemata (auch externe Sichten Views genannt))

Gesamtaufbau



Datenunabhängigkeit

- Prinzip besteht darin, jeden Benutzer bei Veränderungen nur minimale Auswirkungen spüren zu lassen, d.h., Auswirkungen tieferer Ebenen, die ihn/sie im Prinzip auch betreffen würden, sollen nicht weitergegeben werden
- Wir unterscheiden im Folgenden
  - Physische Datenunabhängigkeit
    - Stabilität der konzeptionellen Ebene gegenüber Änderungen der Dateiorganisation oder der angewendeten Zugriffsverfahren
    - D.h. eine Veränderung im internen Schema hat keine Auswirkungen auf die konzeptionelle Ebene
  - Logische Datenunabhängigkeit
    - Stabilität der Benutzungsschnittstelle gegenüber Veränderungen der Datenstrukturen auf der konzeptionellen Ebene
    - D.h. eine Veränderung im konzeptionellen Schema hat keine Auswirkungen auf die externen Sichten
    - Ausnahme können allerdings hier z.B. Löschungen sein
    - Idealerweise werden aber diese fälschlichen Zugriffe ebenfalls abgefangen
    - D.h. z.B. der Zugriff auf nicht mehr vorhandene Attribute wird nicht mehr durchgeführt und löst lediglich eine entsprechende Meldung aus

Datenbank Sprachen

- DDL=Data Definition Language
  - Konzeptionelles Schema wird hierdurch festgelegt
  - Schnittstellen zum Internen Level
- SDL=Storage Definition Language
  - Definiert internes Level
- VDL=View Definition Language
  - Definiert externe Sichten für bestimmte Benutzergruppen
  - Schnittstellen zum konzeptionellen Level
- DML=Data Manipulation Language
  - Dient zur Formulierung von Benutzeranfragen
  - Man unterscheidet
    - Low level prozedural
      - Diese Sprachen greifen nur einzeln auf Einträge in der Datenbank zu (record at a time)
      - Müssen für aufwändigere Operationen in Hochsprachen eingebunden werden
    - High level prozedural
      - Können bereits komplexere Auswertungen (set at a time) (z.B. SQL)
      - Brauchen nicht eingebunden zu werden

Folgerung

- Datenbank Administratoren nutzen spezielle Arten von
  - DDL
  - SDL
  - VDL

- End Benutzer nutzen spezielle Arten von
  - DML

2.1.4 Design Prozess der Datenbank

Phase 1

- Anforderungsanalyse
- Hieraus entstehen Anforderungen an die Daten und ihren Aufbau
- Anforderungen hinsichtlich der Zugriffe

Phase 2

- Entwicklung des konzeptionellen Designs mit Hilfe eines Meta Datenmodells
- Dieses ist DBMS unabhängig
- Zudem gilt, dass das konzeptionelle Design ungleich dem konzeptionellen Schema ist
- DBMS unabhängige Beschreibung der Zugriffe der Anwendungsprogramme

Phase 3

- Wahl des Datenbanksystems (DBMS+DB)
- D.h. Datenbanktyp (Relational, OO, …), Wahl der Query Language, Wahl der zusätzlichen Software

Beachte: Alle bisherigen Phasen (1-3) sind DBMS unabhängig

DBMS abhängige Phasen

Phase 4

- Entwicklung des logischen Designs
- Bestimmung des konzeptionellen und des externen Schemas

Phase 5

- Festlegung des physikalischen Designs
- Ermittlung des internen Schemas

Phase 6

- Implementierung des Systems
- SDL+VDL+DDL Statements für die Festlegung des konzeptionellen, des externen und des internen Schemas
- Für die Anwendungssysteme werden zur Ausgestaltung der Anfrageroutinen Hilfsprogramme in DML + Host Language implementiert

Phase 7

- Laufender Betrieb
- Überwachung & Anpassung
- Performancemessungen & Befragungen von Mitarbeitern

Die Phasen 4-7 sind abhängig von einem gewählten Datenbanksystem

2.2 Datenmodelle

- Ein Datenmodell dient zur Beschreibung von Daten und ihrer Strukturen
- Sie bestimmen
  - Die Syntax (d.h. den Aufbau) und
  - Die Semantik (d.h. die Interpretation)

  von Datenbank Schemata
- Wir vereinbaren

μ: Menge aller möglichen Objekte und Wertebereiche, die die im Schema angegebenen Objekte und Datentypen annehmen können

Σ: Menge von passenden DB Zuständen (d.h. gültigen Ausprägungen)

$\sigma \in \Sigma$ ist ein passender DB Zustand

2.2.1 Ein Meta Modell: Das ER Modell

- Das ER heißt Entity Relationship Model
- Modellierungskonzepte:

Entität

- Eine Entität ist eine „Sache“ die eindeutig identifiziert werden kann und sich von anderen „Sachen“ unterscheidet
- Beispiele hierfür sind eine konkrete Person, ein konkretes Unternehmen, oder ein konkretes Ereignis

Beziehung

- Eine Beziehung ist eine Verknüpfung von Entitäten
- Ein Beispiel hierfür ist die Beziehung Vater-Sohn für zwei konkrete Personen

Entitäten und Entitätstypen

- Ein Entitätstyp ist ein Schema für Entitäten, dieses legt die Struktur von einer Menge von Entitäten (Entitätsmenge) fest
- D.h. die Entitäten haben eine bestimmte Struktur in Form von gemeinsamen Eigenschaften
- Für einen Entitätstyp E gilt allgemein
  - μ(E): Menge aller möglichen Entitäten vom Typ E
  - σ(E): Menge der aktuellen Entitäten vom Typ E im Zustand σ (d.h. $\sigma(E) \subseteq \mu(E)$)
  - Falls also e eine Entität vom Typ E im Zustand σ ist, dann gilt e ist Element von σ(E)
- Beispiele für Entitätstypen sind: Person, Abteilung, Projekt

Beziehungen und Beziehungstypen

- Ein n stelliger Beziehungstyp R von n Entitätstypen $E_1,\ldots,E_n$ ist eine Relation über $E_1,\ldots,E_n$
- Es gilt somit

$$R(E_1,\ldots,E_n) \subseteq E_1 \times E_2 \times \ldots \times E_n$$

d.h. für einen konkreten Zustand σ

$$\sigma(R) \subseteq \sigma(E_1) \times \sigma(E_2) \times \ldots \times \sigma(E_n)$$

- Ein Beziehungstyp ist also ein Schema für Beziehungen, dieser legt die Struktur von einer Menge von Beziehungen (Beziehungsmenge) fest
- Zum Beispiel ist „Ehe“ ein Beziehungstyp zwischen zwei Entitäten des Entitätstyps Person
- Die konkrete Verknüpfung zwischen zwei Entitäten ist dann eine Beziehung
- Damit ist jede konkrete Beziehung Teil der Beziehungsmenge „Ehe“

Attribute, Werte und Wertebereich

- Ein Attribut A ist eine Abbildung von einem dazugehörigen Entitätstyp E in einen dazugehörigen Wertebereich W (bei mehrwertigen in die Potenzmenge von W)
- Beispiel: Das Attribut „Alter“ eines Entitätstypen Person ist eine Abbildung in den Wertebereich „Anzahl von Jahren“
- Mehrere Attribute können in den gleichen Wertebereich W abgebildet werden
  - Die Attribute „Email-Privat“ und „Email-Geschäftlich“ sind Abbildungen in den Wertebereich „Email“
- Statt $A_1:E\rightarrow W_1$, …, $A_n:E\rightarrow W_n$
  schreiben wir einfach $E(A_1,\ldots,A_n)$
- Die Wertebereiche $W_1,\ldots,W_n$ sind dabei entweder
  - Mengen
  - Potenzmengen oder
  - Kartesische Produkte von Mengen und/oder Potenzmengen

Identifizierung von Entitäten – Definitionen

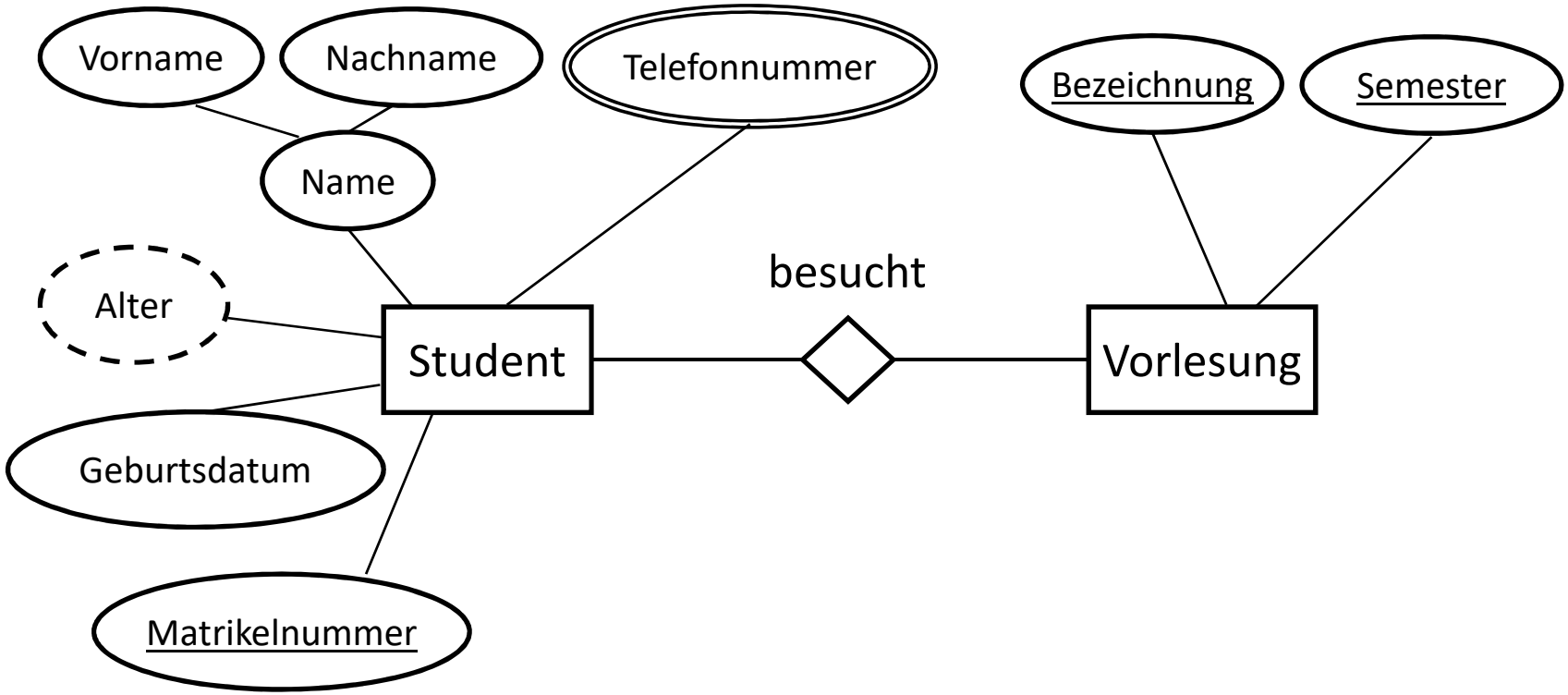
- Ein *Superschlüssel* eines Entitätstyps ist eine Menge von einem oder mehreren Attributen, die eine Entität eindeutig identifizieren
- Ein *Schlüsselkandidat* oder kurz *Schlüssel* ist ein minimaler Superschlüssel

  Achtung: Die genaue Definition von minimal erfolgt später
- Jedes Attribut, das Teil eines Schlüsselkandidaten ist, heißt *Schlüsselattribut*, ansonsten Nichtschlüsselattribut
- Für jeden Entitätstyp wird genau ein Schlüsselkandidat ausgewählt. Dieser wird als *Primärschlüssel* bezeichnet
- Damit gilt: Ein Entitätstyp kann Schlüsselattribute besitzen, die nicht Teil des Primärschlüssels sind

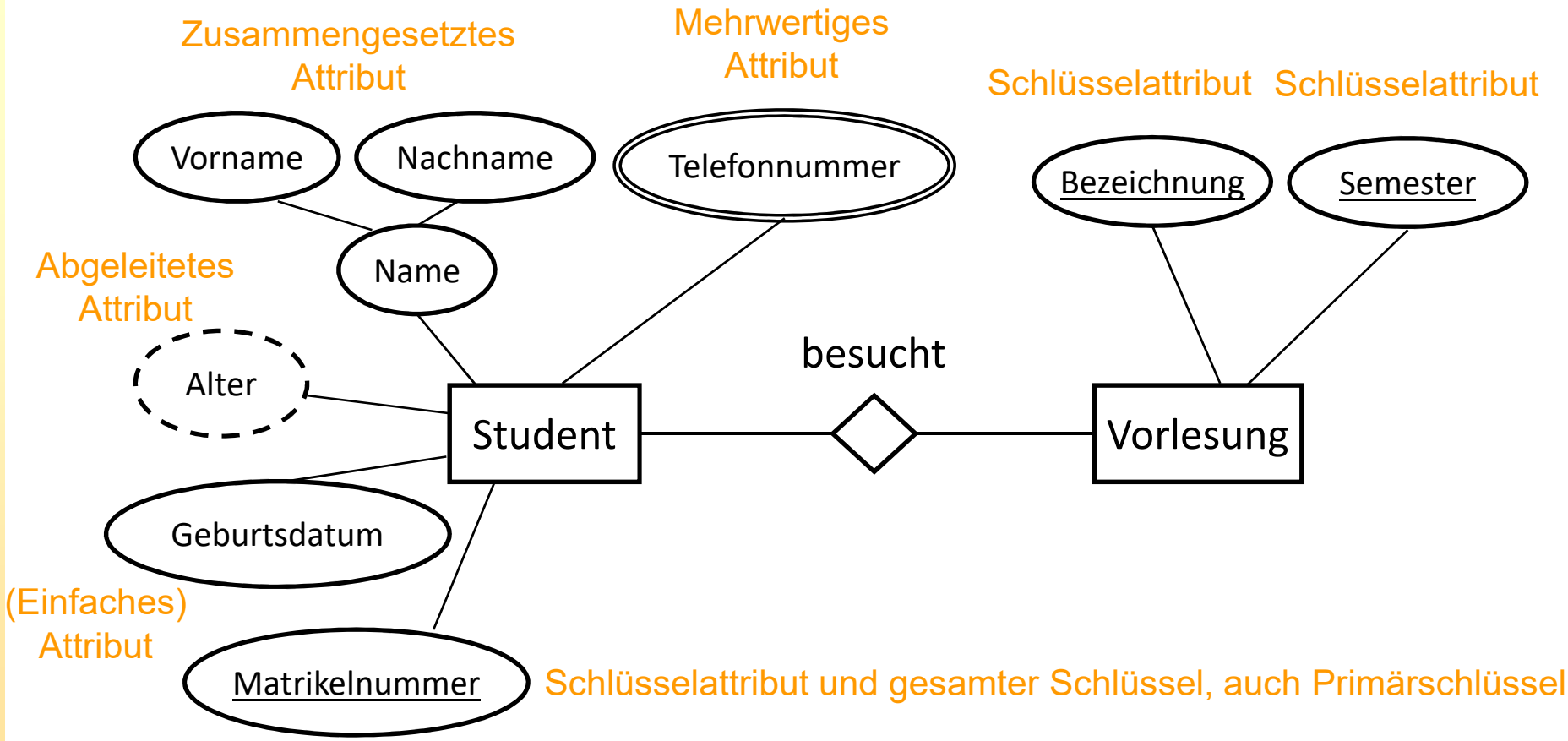
Modellierung von Attributen

- Ein Entitätstyp E kann einen Primärschlüssel besitzen, welcher einen unterschiedlichen Wert für alle Entitäten vom Typ E besitzt und somit das einzelne Objekt eindeutig identifiziert
- Der Primärschlüssel besteht aus einem oder mehreren Schlüsselattributen
- Man unterscheidet weiterhin in
  - Einfache/zusammengesetzte Attribute
    - Einfach: Wohnort, Hausnummer
    - Zusammengesetzt: Adresse (bestehend aus Straße, Ort, Land, …)
  - Einwertige/mehrwertige Attribute
    - Einwertig: Matrikelnummer
    - Mehrwertig: Telefonnummern (Wenn beliebig viele eingetragen werden dürfen)
  - Gespeicherte/ableitbare Attribute
    - Gespeichert: Geburtsdatum
    - Ableitbar: Alter (abgeleitet aus Geburtsdatum und aktuellem Datum)

Beispiel für verschiedene Attribute



Beispiel



Falls die Attribute Vorname, Name, Geburtstag zusammen einen Schlüssel bilden, wären auch diese Attribute Schlüsselattribute, obwohl sie nicht zum Primärschlüssel gehören

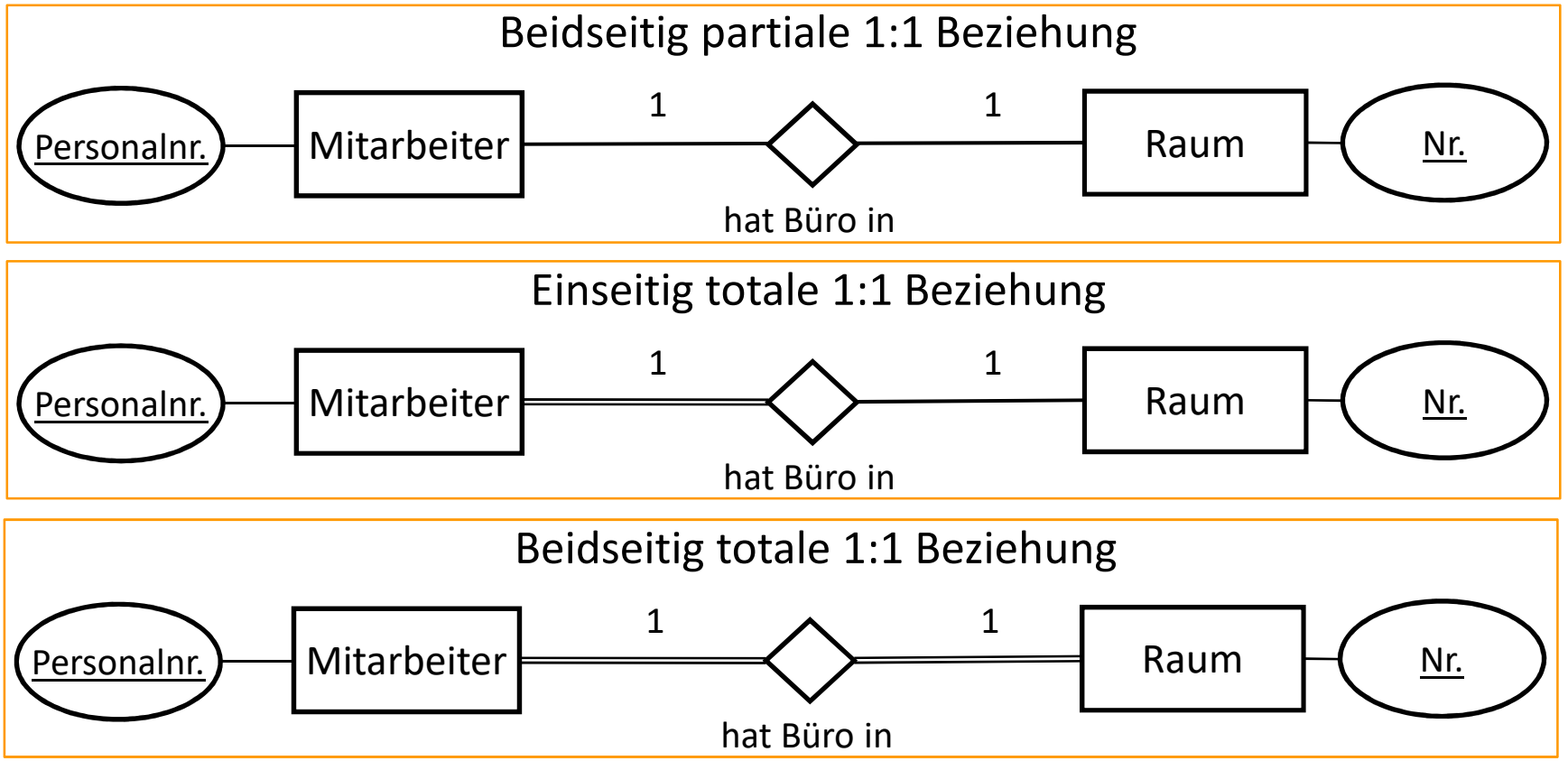
Darstellung der Kardinalität

- Die Kardinalität gibt an, wie oft eine Entität eines Entitätstypen mit einem Beziehungstypen **maximal** verknüpft werden kann
- Dabei gibt es nur die Werte „1“ oder „N“ bzw. „M“
- „1“ steht dafür, dass jedes einzelne Element nur in maximal einer Beziehung des Beziehungstyps auftauchen kann
- „N“ bzw. „M“ stehen dafür, dass jedes einzelne Element in beliebig vielen Beziehungen des Beziehungstyps auftauchen kann
- Die Kardinalität regelt also die **Maximalzahl des Vorkommens** eines Elementes in einem Beziehungstyp
- Die Kardinalität wird jeweils an den Linien zum Beziehungstyp direkt angegeben

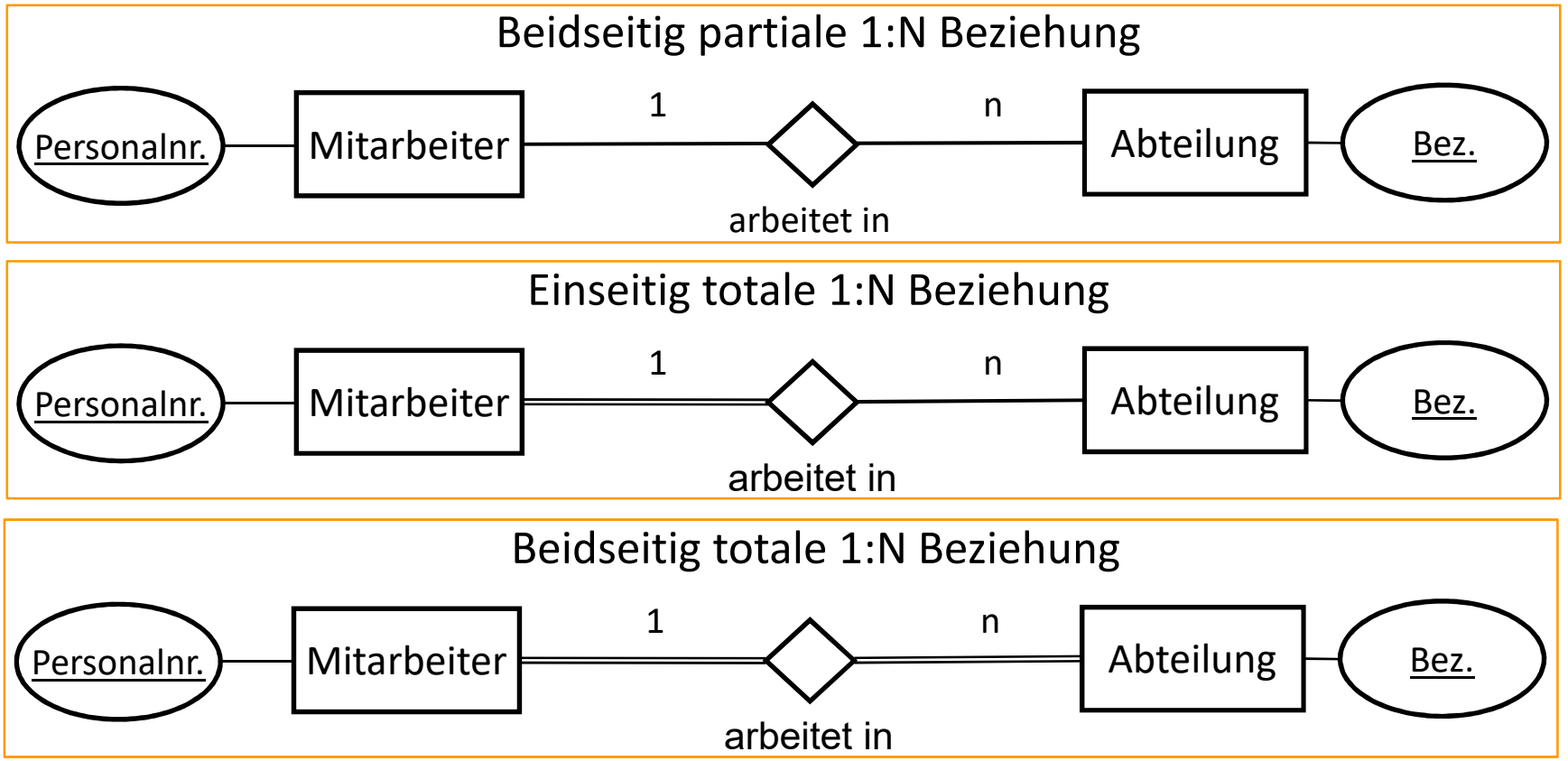
Darstellung der Partizipation

- Die Partizipation gibt an, wie oft eine Entität eines Entitätstypen mit einem Beziehungstypen **minimal** verknüpft werden muss
- Dabei unterscheiden wir zwei Fälle:
  - Partiale Partizipation: Eine Entität **kann** in einer Beziehung dieses Typs auftauchen
  - Totale Partizipation: Eine Entität **muss** in einer Beziehung dieses Typs auftauchen
- Die Partizipation definiert die **Minimalzahl des Vorkommens** eines Elementes in einem Beziehungstyp
- Eine Partiale Partizipation wird durch eine *einfache Linie* vom Entitätstyp zum Beziehungstyp gekennzeichnet
- Eine Totale Partizipation wird durch eine *doppelte Linie* vom Entitätstyp zum Beziehungstyp gekennzeichnet

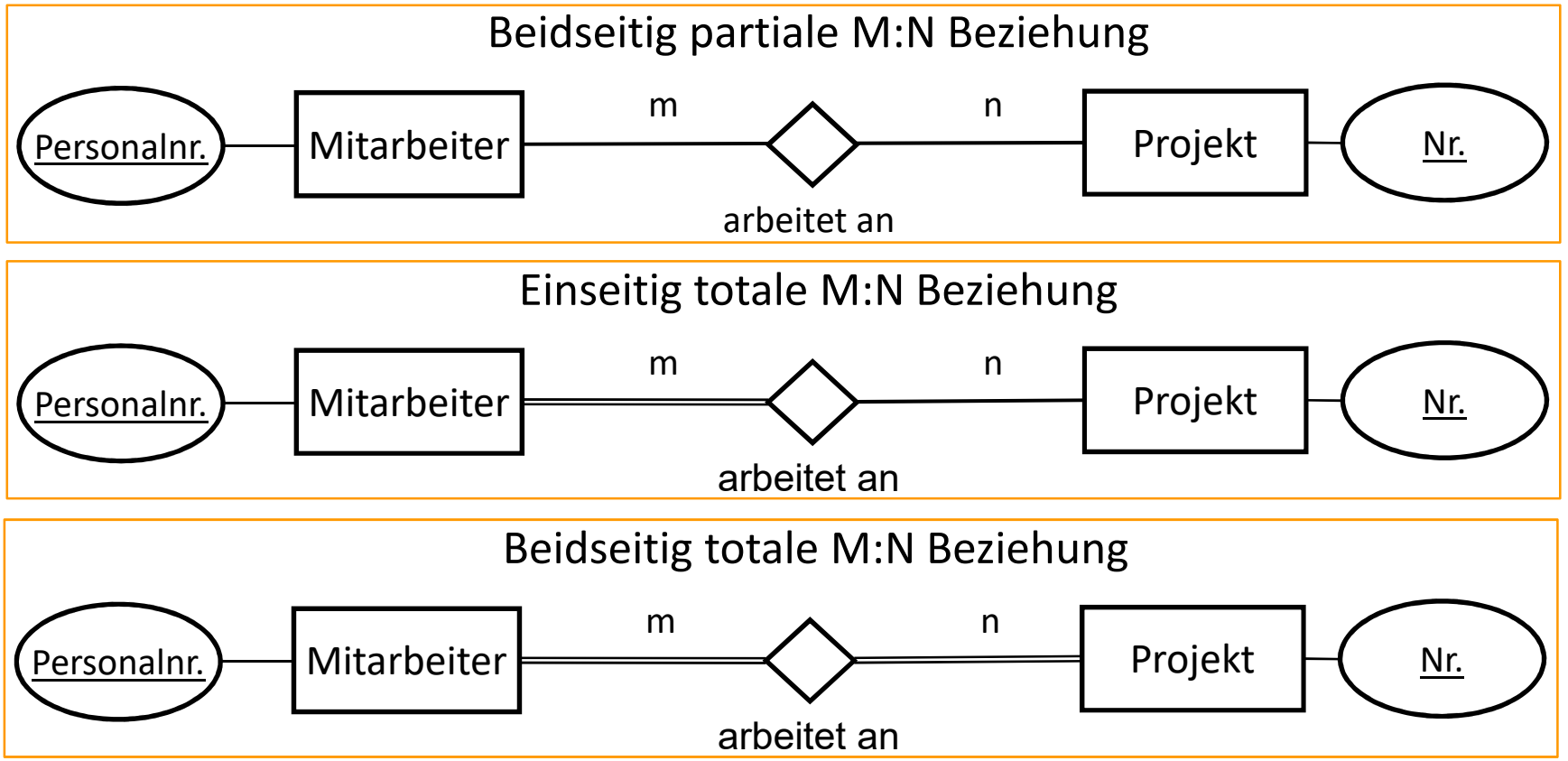
Beispiele für Kardinalität und Partizipation



Beispiele für Kardinalität und Partizipation



Beispiele für Kardinalität und Partizipation



Starke und Schwache Entitätstypen

- (Starke) Entitätstypen
  - besitzen ein oder mehrere Schlüsselattribut(e) das/die den Primärschlüssel bildet/bilden
  - Jede konkrete Entität ist also eindeutig durch die Werte identifizierbar.
- Schwache Entitätstypen
  - Sind Entitätstypen die nicht vollständig über eigene Schlüsselattribute identifiziert werden können oder gar keine Schlüsselattribute besitzen
  - Schlüsselattribute können zusammen mit Schlüsselattributen weiterer Entitäten den Primärschlüssel bilden
  - Diese(r) andere Entitätstyp(en) wird/werden als Besitzer charakterisiert
  - Schlüsselattribute eines Schwachen Entitätstypen werden durch eine gestrichelte Linie gesondert kenntlich gemacht

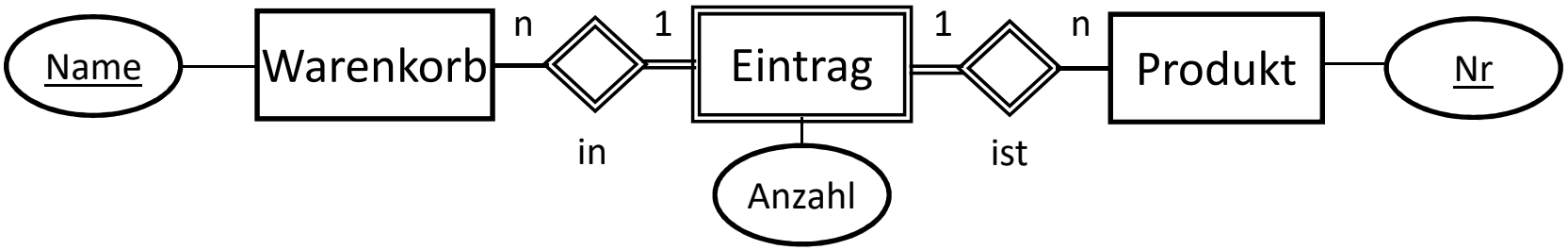
Identifizierende Beziehungstypen

- Schwache Entitätstypen
  - Es existiert eine totale Partizipation dieses Entitätstyps über einen oder mehrere andere Entitätstypen
  - Sofern der Schwache Entitätstyp kein Schlüsselattribut besitzt können schwache Entitäten des selben Typs nicht der gleichen Entität zugeordnet werden

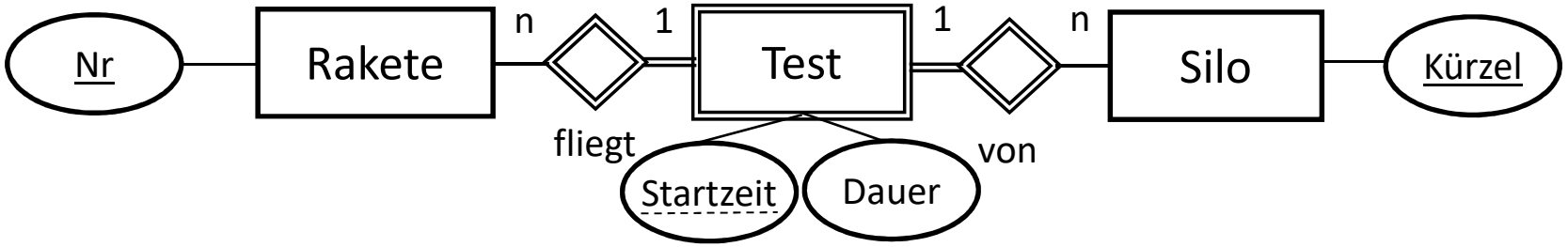
- Identifizierender Beziehungstyp
  - Macht kenntlich, dass der Schwache Entitätstyp über den Primärschlüssel dieses Entitätstypen (mit) identifiziert wird
  - Es existiert immer eine 1:X Beziehung, die auf der Seite des schwachen Entitätstypens total ist

Beispiele für schwache Entitätstypen

Modell A: Schwacher Entitätstyp **ohne** weiteres Schlüsselattribut



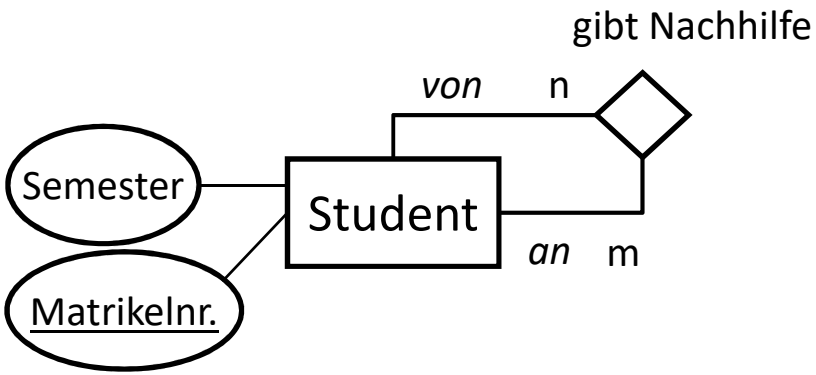
Modell B: Schwacher Entitätstyp **mit** weiterem Schlüsselattribut



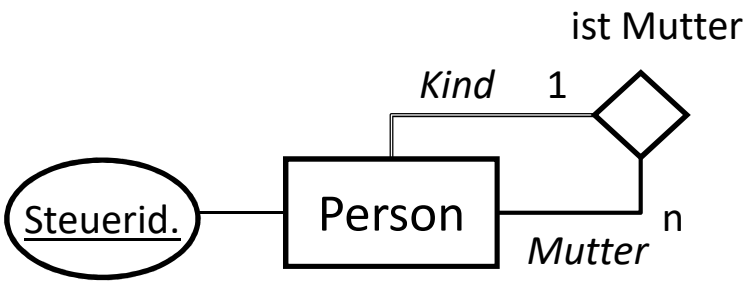
Rekursive Beziehungstypen

- Ein Entitätstyp kann mehrmals mit dem selben Beziehungstypen verbunden sein
- Je nach Anzahl *x* der Verbindungen stehen also *x* Entitäten dieses Entitätstyps in jeder Beziehung dieses Beziehungstyps
- An den Linien kann zusätzlich eingetragen werden welche Rolle eine Entität in dieser Beziehung übernimmt

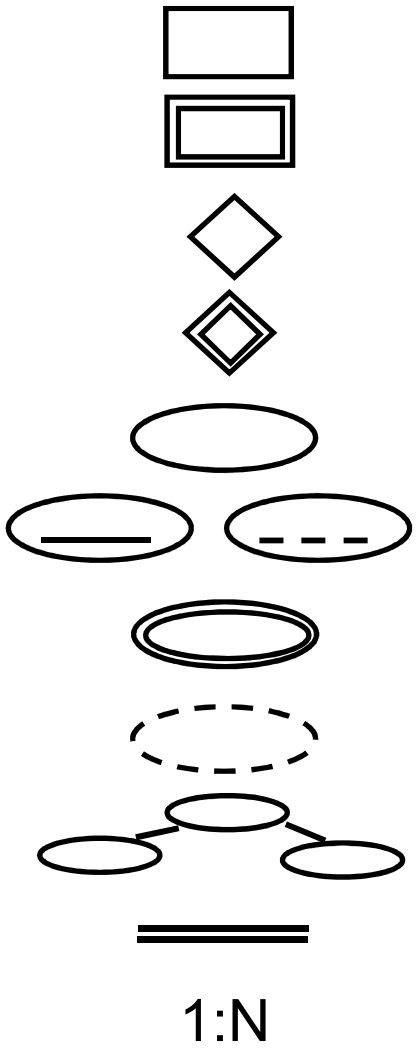
Modell A



Modell B



Darstellungen



- Entitätstyp
- Schwacher Entitätstyp
- Beziehungstyp
- Identifizierender Beziehungstyp
- Attribut
- Schlüsselattribut
- Mehrwertiges Attribut
- Abgeleitetes Attribut
- Zusammengesetztes Attribut
- Totalität
- Kardinalität

2.2.2 Das Relationale Modell

- Ist ein klassisches Datenmodell, das direkt in Datenbank Schemata eingegangen ist
- Besitzt eine sehr einfache Struktur
- Das **Instrument der Relation** stellt die **Grundlage dieses Modells** dar
- Ein Wertebereich (Domäne) ist eine Menge von atomaren Werten
  - d.h. unteilbar in der Datenbankwelt
  - Man definiert sie mit Datentypen und Formatierungsvorschriften
- Beispiel
  - Telefonnummern in den USA
  - (Staaten Code) Zulässige Durchwahl
  - (ddd) ddd dddd, mit d (digit aus {0,1,2,3,4,5,6,7,8,9})

Relationales Schema

- Ein relationales Schema R, geschrieben als $R(A_1,\ldots,A_n)$ (mit R Relationsname und den Attributen $A_1,\ldots,A_n$) beschreibt die Relation R
- Jedes Attribut A_i besitzt eine Domäne $dom(A_i)$, die den möglichen Wertebereich festlegt
- Damit kann der Fall eintreten, dass zwei Domänen identisch sind, d.h. es gilt $dom(A_i)=dom(A_j)$

  Beispiel hierfür wären Telefonnummer Büro und Telefonnummer privat
- Eine Relation r als Instanz eines Relationsschema $R(A_1,\ldots,A_n)$ (auch geschrieben als r(R)) ist eine Menge von n-Tupeln
- Damit gilt

$$r(R)=\left\{(t_1,\ldots,t_n)\mid \forall i\in\{1,\ldots,n\}: t_i\in dom(A_i)\vee t_i=NULL\right\}$$
$$\Rightarrow$$
$$r(R)\subseteq dom(A_1)\times\ldots\times dom(A_i)\times\ldots\times dom(A_n)$$

Folgerungen

- Somit gilt für einen aktuellen Relationszustand

$$\sigma(R) \subseteq \mu(R) = dom(A_1) \times \ldots \times dom(A_i) \times \ldots \times dom(A_n)$$

- Offensichtlich beinhaltet σ nur zulässige n-Tupel der Relation
- Im Gegensatz dazu umfasst μ alle theoretisch möglichen n-Tupel, die über den Domänen der betrachteten Relation gebildet werden können
- Offensichtlich gibt es für die einzelnen Tupel einer Relation **keine Ordnung**, d.h. die Reihenfolge ist per Definition unbestimmt
- Informationstechnisch ist das natürlich nicht der Fall, da die Daten immer in einer bestimmten Reihenfolge abgelegt sind

Wichtige Eigenschaften

- Demgegenüber gibt es in jedem Tupel eine Ordnung der einzelnen Attribute, d.h. in jeder Instanz eine Ordnung der Werte
- Man könnte dies offensichtlich umgehen, indem man ein relationales Schema nicht als Tupel sondern als Menge von Attributen festlegt
- Wichtig
  - In Relationen tauchen nur atomare (d.h. unteilbare) Werte auf
  - Dies muss durch die jeweiligen Domänen gewährleistet werden
  - Dies bedeutet insbesondere, dass es keine mehrwertigen oder zusammengesetzten Attribute in Relationen gibt
  - Somit können wir nicht eine Attributsdomäne als eine Menge von Domänen definieren
  - Das Relationale Modell gilt daher als „flach“

Formale Schreibweisen

- Wir vereinbaren

1. $R(A_1,...,A_n)$: Relationales Schema der Stelligkeit (des Grades) n
2. $t \in r(R), t = (v_1,...,v_n), v_i$ gehört zu Attribut A_i. Dabei lassen sich die einzelnen Komponenten durch $t[A_i]$ ansprechen. Ein Sub-Tupel kann durch $t[A_v,...,A_z]$ angesprochen werden
3. Q, R, S sind Relationennamen
4. q, r, s sind Zustände von Relationen
5. t, u, v sind Tupel von Relationen
6. Der Relationenname R alleine bezieht sich auf den aktuellen Zustand der Relation während $R(A_1,...,A_n)$ das relationale Schema bezeichnet
7. Attribute (also ihre Domäne) sind ansprechbar über $R.A_i$

Regeln für das Relationale Modell

- Domänen Regel
  - Domänen sind ausschließlich über atomare Typen zu definieren
  - Siehe dazu die Regeln für SQL Datentypen
- Schlüssel Regel
  - Eine Relation ist eine Menge von paarweise unterschiedlichen Tupeln
  - Dazu gilt

Zwei Tupel t_1, t_2 mit

$$t_1 = (t_{1,1}, ..., t_{1,n}), t_2 = (t_{2,1}, ..., t_{2,n}) \in dom(A_1) \times ... \times dom(A_i) \times ... \times dom(A_n)$$

heißen identisch

$$\Leftrightarrow \forall i \in \{1, ..., n\} : t_{1,i} = t_{2,i}$$

Nicht identische Tupel

- Des weiteren gilt

Falls gilt für $t_1 = \left(t_{1,1},\ldots,t_{1,n}\right), t_2 = \left(t_{2,1},\ldots,t_{2,n}\right)$ und $S = \left\{A_{k_1},\ldots,A_{k_l}\right\}$,

mit $l \le n$ und $\forall j \in \left\{1,\ldots,l\right\} : 1 \le k_j \le n$:

$\exists A_d \in S = \left\{A_{k_1},\ldots,A_{k_l}\right\} : t_{1,d} \neq t_{2,d} \Rightarrow t_1 = \left(t_{1,1},\ldots,t_{1,n}\right)$ und $t_2 = \left(t_{2,1},\ldots,t_{2,n}\right)$

sind nicht identisch

Man schreibt in diesem Fall kurz: $t_1\left[S\right] \neq t_2\left[S\right]$

Superschlüssel (Super Key)

- Eine Menge von Attributen S heißt für ein relationales Schema $R(A_1,\ldots,A_n)$ Superschlüssel oder Super Key, genau dann wenn S Teilmenge von $\{A_1,\ldots,A_n\}$ ist und es gilt zudem

$$\forall t_1 = \left(t_{1,1},\ldots,t_{1,n}\right) \in R : \forall t_2 = \left(t_{2,1},\ldots,t_{2,n}\right) \in R : t_1\left[S\right] \neq t_2\left[S\right]$$

- Ein Schlüssel S eines relationalen Schemas $R(A_1,\ldots,A_n)$ ist ein Superschlüssel mit der folgenden zusätzlichen Eigenschaft

$$\text{Falls } S = \left\{A_{k_1},\ldots,A_{k_l}\right\} : \forall c \in \left\{1,\ldots,l\right\} : S / \left\{A_{k_c}\right\} \text{ ist kein Superschlüssel } \left(\text{Super Key}\right)$$

Begriffsabgrenzung: Schlüssel

- Eine Attributsmenge heißt Superschlüssel, wenn sie eine Relation eindeutig identifiziert
- Eine Attributsmenge heißt Schlüssel, wenn sie ein Superschlüssel und minimal ist
- *Achtung: Es kann mehrere (Super-)Schlüssel in einer Relation geben*
- Alle Schlüssel einer Relation werden auch als Kandidatenschlüssel bezeichnet
- Ein ausgewählter Kandidatenschlüssel wird als Primärschlüssel bezeichnet

Konsequenz

- Jedes relationale Schema $R(A_1,\ldots,A_n)$ hat einen Superschlüssel und einen Schlüsselkandidaten
- Dies ist im Extremfall die gesamte Menge der vorhandenen Attribute $\{A_1,\ldots,A_n\}$
- Jeder Schlüssel kann als Primärschlüssel eingesetzt werden
- Alle Attribute die Teil des gewählten Primärschlüssels sind, sind im relationalen Schema zu unterstreichen und somit kenntlich zu machen

Integritätsregeln

- Entitätsintegrität
  - Hier wird gefordert, dass niemals in einer Relation ein Tupel existiert, das den Wert NULL in dem Attribut hat, welches den Primärschlüssel darstellt
  - Nur so bleibt die Primärschlüsseleigenschaft erhalten
- Schlüsselintegrität

  Alle Tupel sind unterschiedlich bezüglich eines angegebenen Schlüssels
- Referentielle Integrität
  - Wir betrachten zwei Relationen R_1 und R_2
  - Es sei FK eine Attributsmenge der Relation R_1 und
  - PK sei der Primärschlüssel von R_2
  - Für alle diese Fälle verlangt die referentielle Integrität, dass gilt

$$\forall t_1 = \left(t_{1,1}, \ldots, t_{1,n}\right) \in R_1 : \exists t_2 = \left(t_{2,1}, \ldots, t_{2,n}\right) \in R_2 :$$
$$t_1\left[FK\right] = t_2\left[PK\right] \vee t_1\left[FK\right] = NULL$$

Interpretation

- Was bedeutet somit referentielle Integrität?
  - Man stellt über eine Menge von Attributen in einer Relation eine Verbindung zu einer anderen Relation her
  - Dies geschieht durch einen Primärschlüssel PK, der eindeutig jedes Tupel in der anderen Relation identifiziert
  - Somit muss ein Verweis auf diese zusätzlichen „Stammdaten“ möglich sein
  - Dies gelingt allerdings nur über die referentielle Integritätsbedingung
- FK wird daher auch als **Fremdschlüssel** (Foreign Key) in der Relation R_1 bezeichnet
- Ein Fremdschlüssel ist somit eine Menge von Attributen in einer Relation, die zusammen einen Primärschlüssel einer anderen Relation definieren

Beispiel

- Relation 1 (Schulklasse)

Schulklasse			
Bezeichnung (Teil des PK)	**Name**	**Vorname**	**Geburtsdatum**
10a – 4711	Rech	Kurt	23.03.2001
10a – 4711	Kranich	Helga	22.04.2001
…	…	…	…
…	…	…	…

- Relation 2 (Einwohner der Stadt XX)

Einwohner der Stadt XX			
Name (PK)	**Vorname (PK)**	**Geburtsdatum (PK)**	**Adresse**
Rech	Kurt	23.03.2001	Willistraße 12
Kranich	Helga	22.04.2001	Auf der Howe 23
…	…	…	…
…	…	…	…
…	…	…	…

Beachte

- Diese Integritätsbedingungen werden direkt im Datenbankschema festgehalten
- Auf diese Weise wird das System ihre Einhaltung in allen Instanzen sicherstellen
- Dies wird innerhalb der DDL festgelegt
- Zudem kann es auch semantische Integritätsbedingungen geben
  - z.B. Gehälter von Auszubildenden sind immer geringer als von Ausgelernten

Konsequenz für Updates

- Die Operation Insert kann verletzen
  - Schlüsselintegrität
  - Bereichsintegrität
  - Referenzielle Integrität
- Die Operation Delete kann verletzen
  - Referenzielle Integrität
- Die Operation Modify kann verletzen
  - Schlüsselintegrität
  - Bereichsintegrität
  - Referenzielle Integrität
- Das System muss auf auftretende Verletzungen reagieren. Zum Beispiel durch
  - Zurückweisen mit Erklärungen
  - Anpassungsvorschlägen ohne deren Annahme die Transaktion nicht zu Ende durchgeführt wird

Damit bestimmt die DDL

- Die Relationen der Datenbank
- Deren Namen und Attribute
- Die Domänen der Attribute und deren Datentypen
- Die Schlüssel, d.h.
  - Die Primärschlüssel
  - Die Fremdschlüssel

Konvertierungen in das relationale Modell

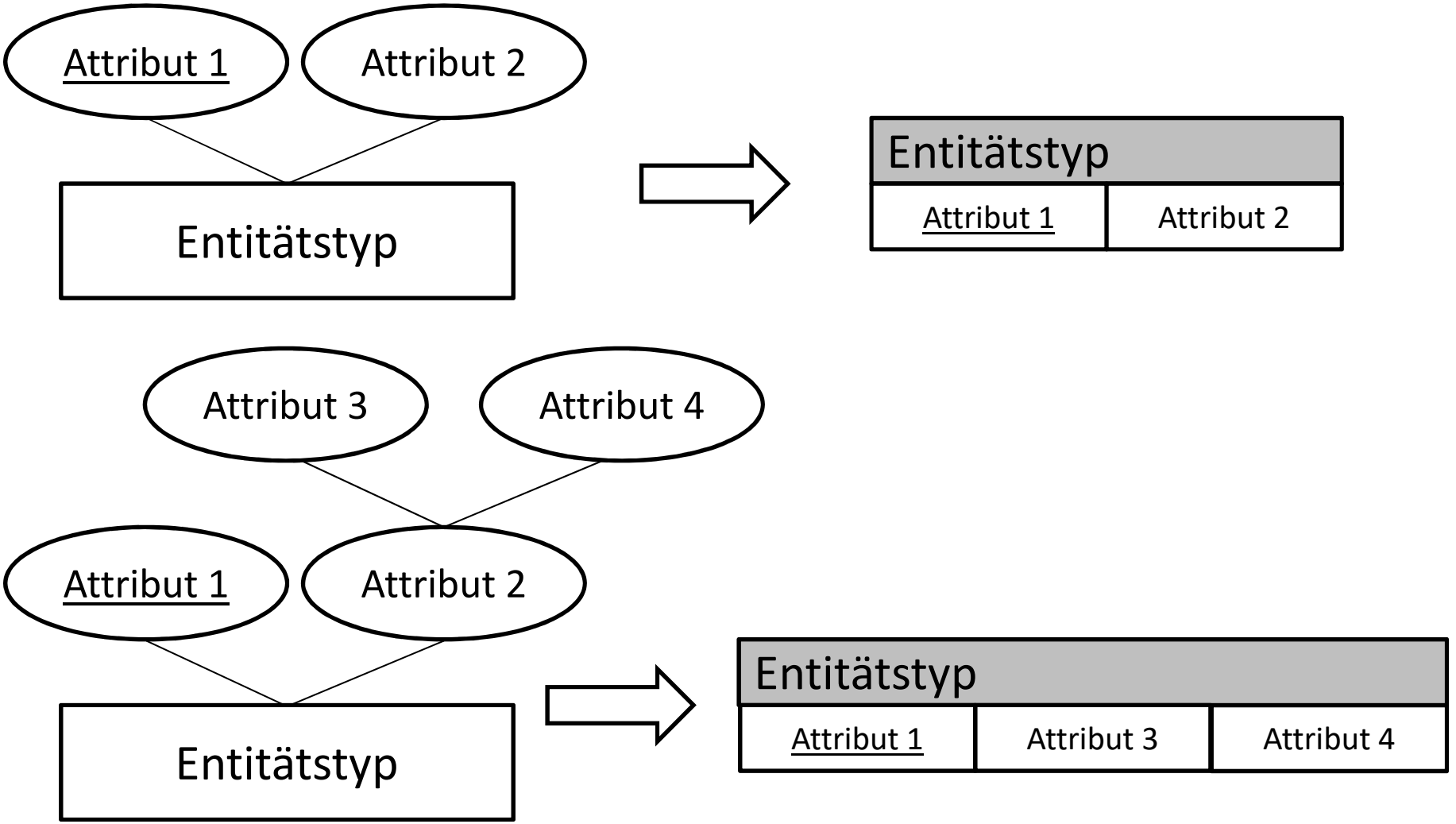
- Meist liegt zunächst ein Datenbankentwurf in einem leistungsfähigen Meta Modell vor
- Dies kann zum Beispiel
  - ein ER Modell oder
  - ein EER Modell sein
- Offensichtlich entspricht dies noch nicht den Standards, die in einem relationalen Modell gelten müssen
- Daher muss – falls eine relationale Datenbank zum Einsatz kommen soll – eine betrachtete Instanz dieses Modells noch in ein relationales Schema umgesetzt werden
- Wir betrachten den Konvertierungsprozess vom ER Modell in das relationale Modell

Vom ER Modell zum Relationalen Modell

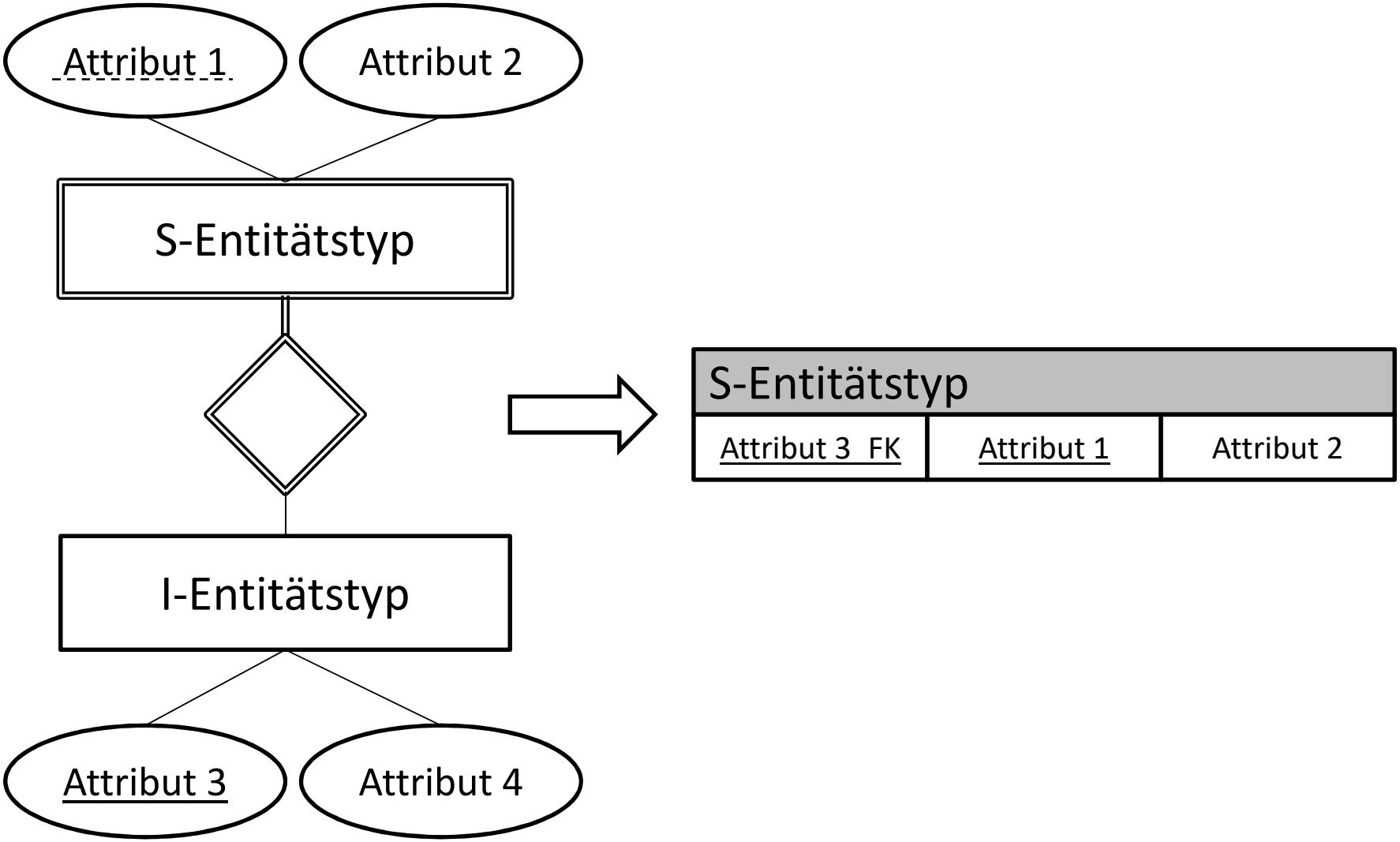
- **Schritt 1**
  - Für jeden starken (d.h. nicht schwachen) Entitätstypen wird eine eigene Relation konstruiert, diese trägt dann die Bezeichnung des Entitätstypen
  - Dabei werden die einfachen eindeutigen Attribute (auch die der zusammengesetzten eindeutigen Attribute) eingetragen und die Attribute des Primärschlüssels unterstrichen

- **Schritt 2**
  - Für jeden schwachen Entitätstypen W mit besitzenden Entitätstypen (E_1, …, E_N) bilde Relation R mit allen einfachen eindeutigen Attributen von W
  - und den Schlüsseln von (E_1, …, E_N) als Fremdschlüssel
  - Der Primärschlüssel besteht dann aus dem Schlüsselattributen von W (falls vorhanden) und den als Fremdschlüssel kenntlich gemachten Attributen der besitzenden Entitätstypen

Schritt 1: Starker Entitätstyp



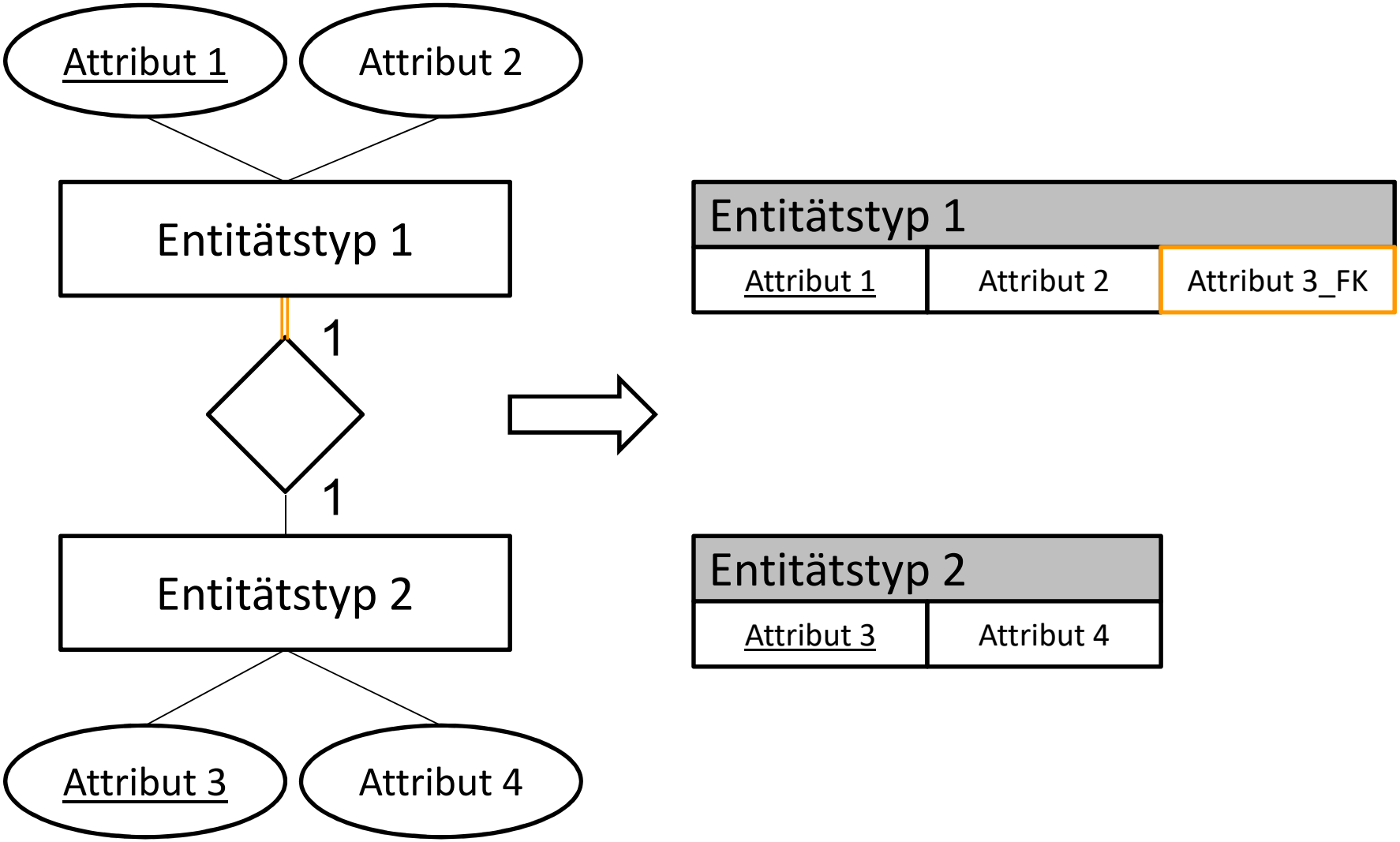
Schritt 2: Schwacher Entitätstyp



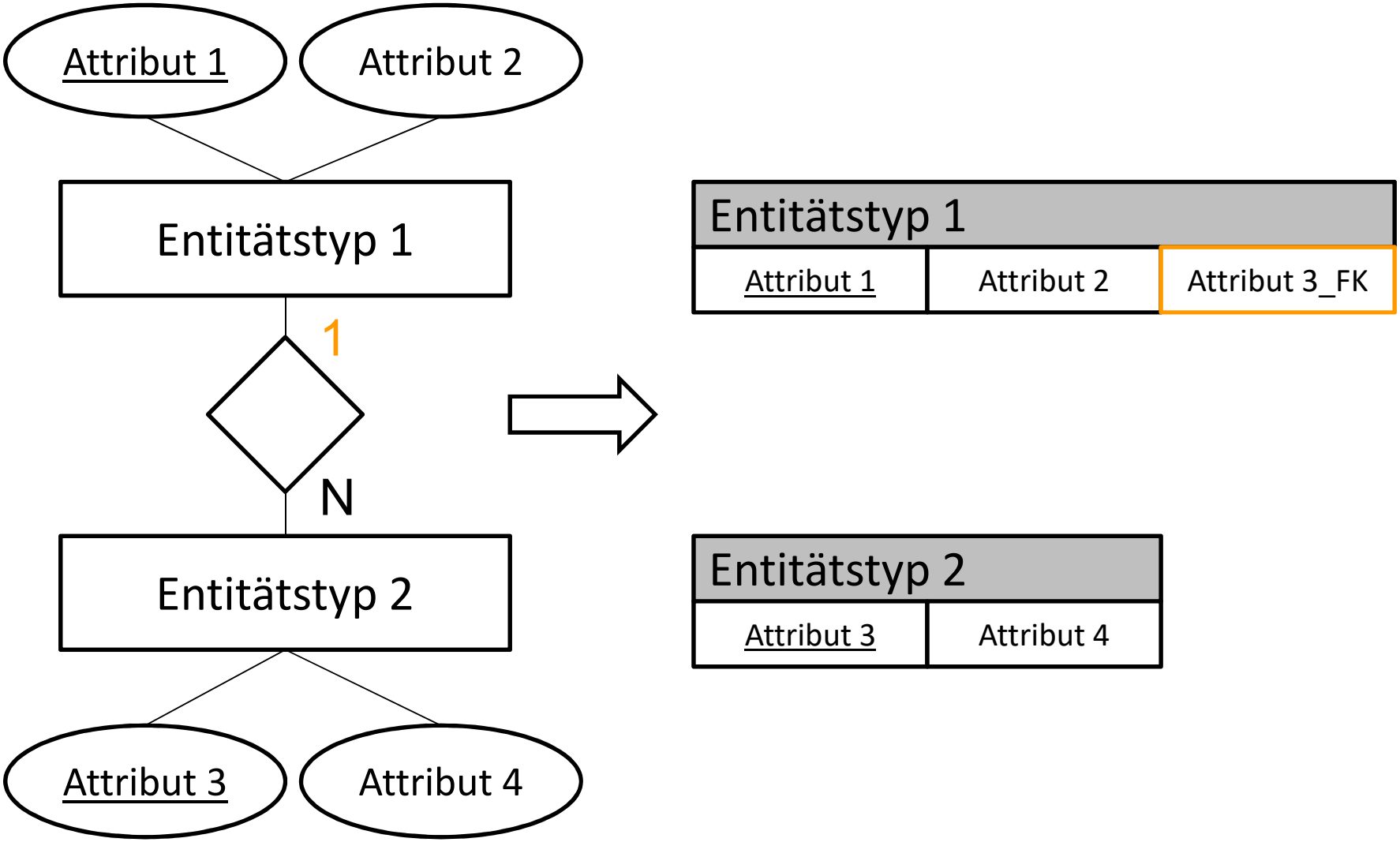
Vom ER Modell zum Relationalen Modell

- *Schritt 3 und 4 erweitern nun die in Schritt 1 und 2 erzeugten Relationen.*
- *Die identifizierenden Beziehungstypen aus Schritt 2 werden nicht erneut betrachtet*
- **Schritt 3**
  - Für jede 1:1 Beziehung, trage – falls vorhanden – in die Relation der totalen Seite die Schlüsselattribute des anderen Entitätstypen als Fremdschlüssel ein
  - Falls keine totale Seite vorhanden ist, nehme eine beliebige von beiden und trage in die Relation dieses Entitätstypen die Schlüsselattribute des anderen Entitätstypen als Fremdschlüssel ein
- **Schritt 4**
  - Für jede 1:N Beziehung, trage in die Relation der eindeutigen Seite (1:) alle Attribute des Primärschlüssels der nicht eindeutigen Seite (N:) als Fremdschlüssel ein

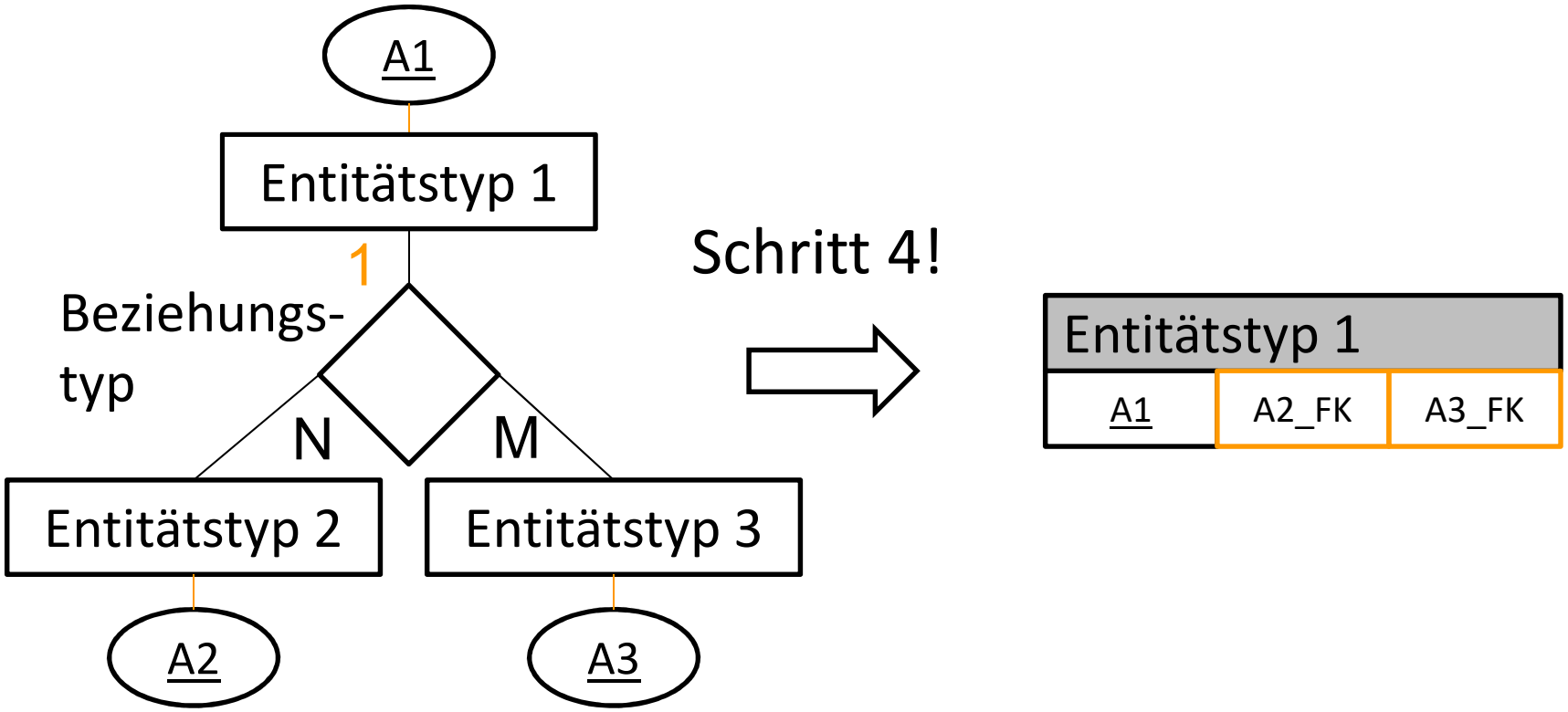
Schritt 3: 1:1 Beziehungen



Schritt 4: 1:N Beziehungen



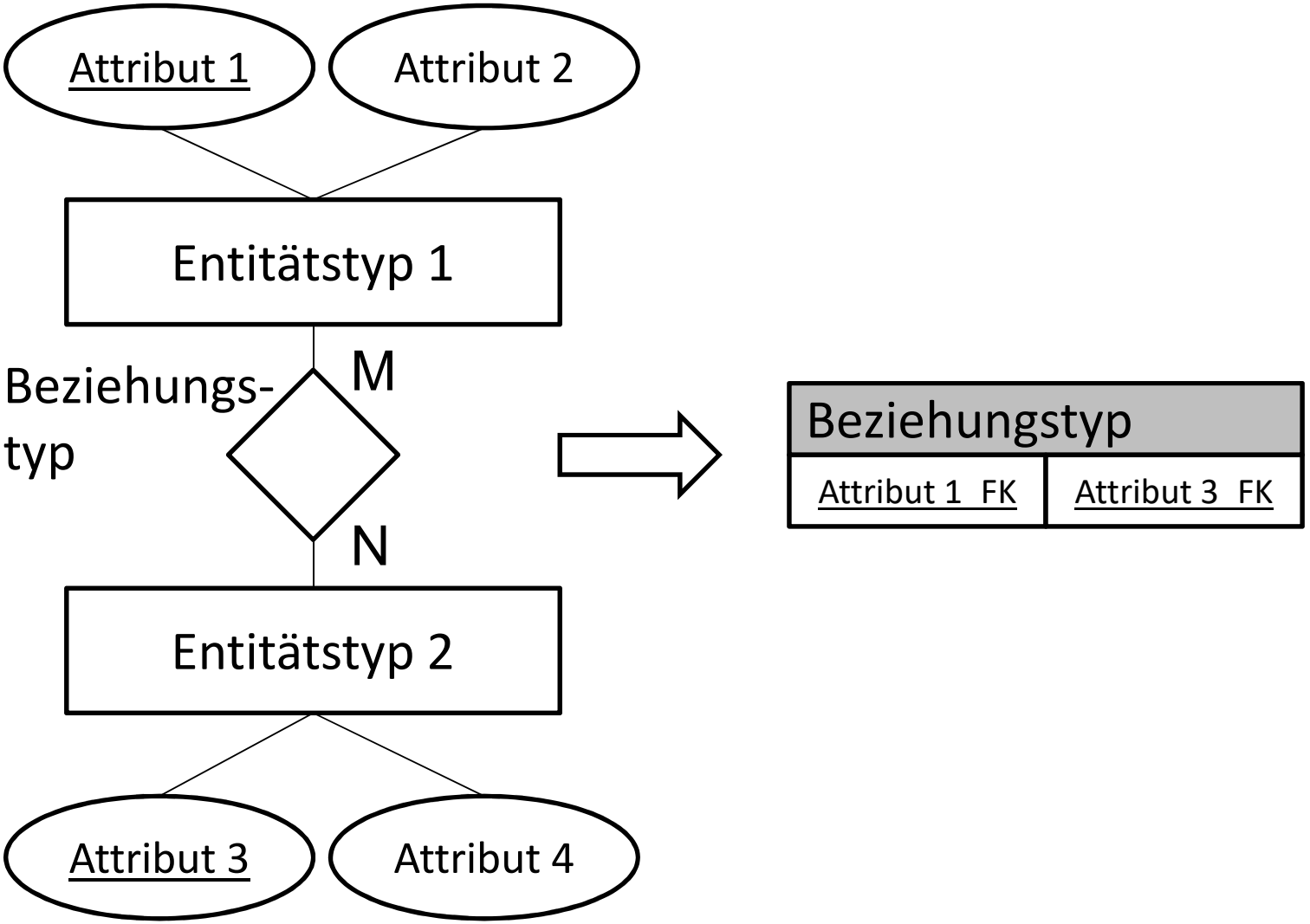
Schritt 4: Beziehungstypen mit mehr als zwei beteiligten Entitätstypen und „1“-Kardinalität



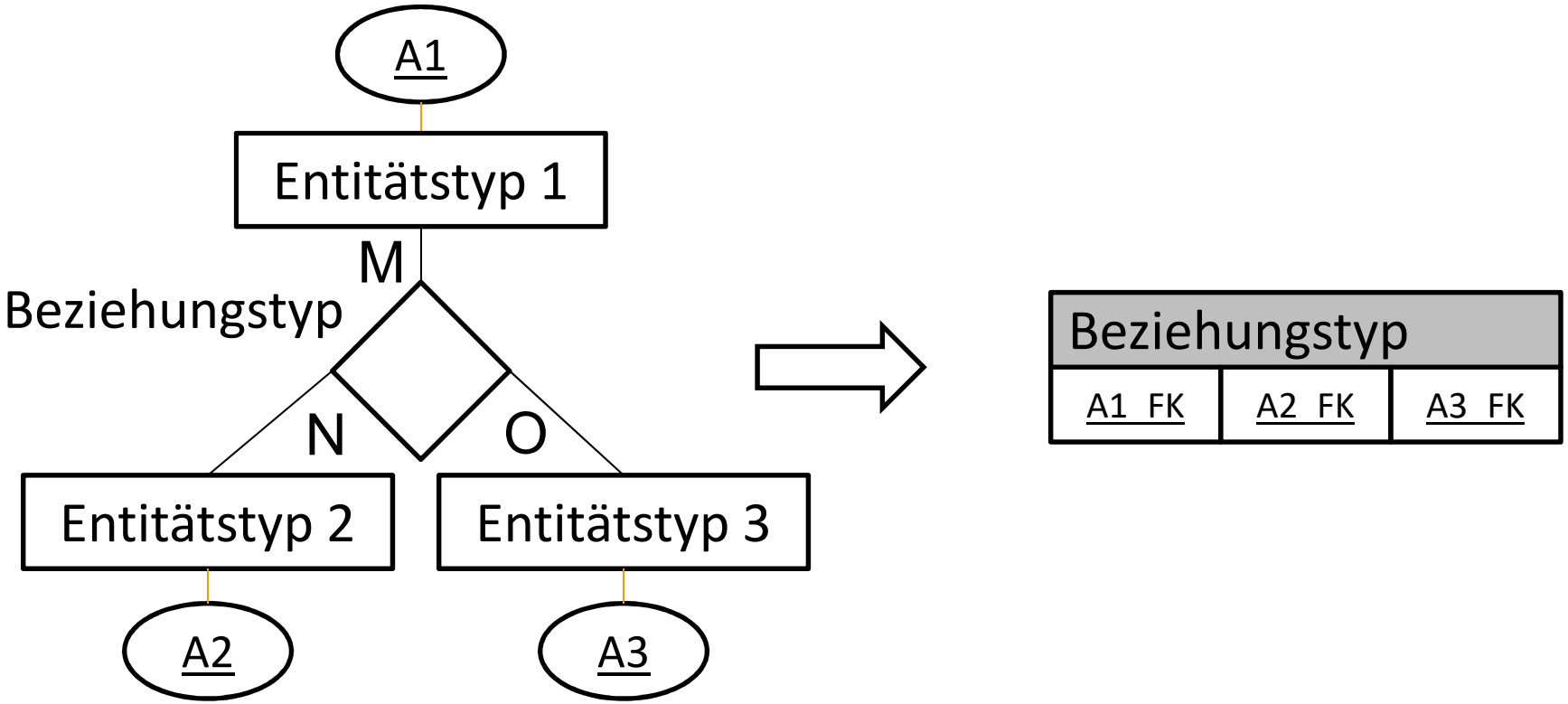
Vom ER Modell zum Relationalen Modell

- **Schritt 5**
  - Für jede M:N Beziehung bilde eine neue Relation R mit einer sinnvollen Bezeichnung
  - Trage in diese die Primärschlüssel aus beiden Relationen als Fremdschlüssel ein und markiere diese als Primärschlüssel der Relation R
  - Dasselbe geschieht auch für alle Beziehungstypen mit mehr als zwei beteiligten Entitätstypen (auch bei Rekursiven Beziehungstypen; wähle hier für die Primärschlüsselattribute unterschiedliche Bezeichnungen)
- **Schritt 6**
  - Sei A ein mehrwertiges Attribut von Entitätstyp E
  - Dann bilde eine neue Relation mit dem Primärschlüssel von E als Fremdschlüssel und das mehrwertige Attribut als zusätzlich Attribut der neuen Relation
  - Aus allen Attributen wird dann der Primärschlüssel der neuen Relation gebildet

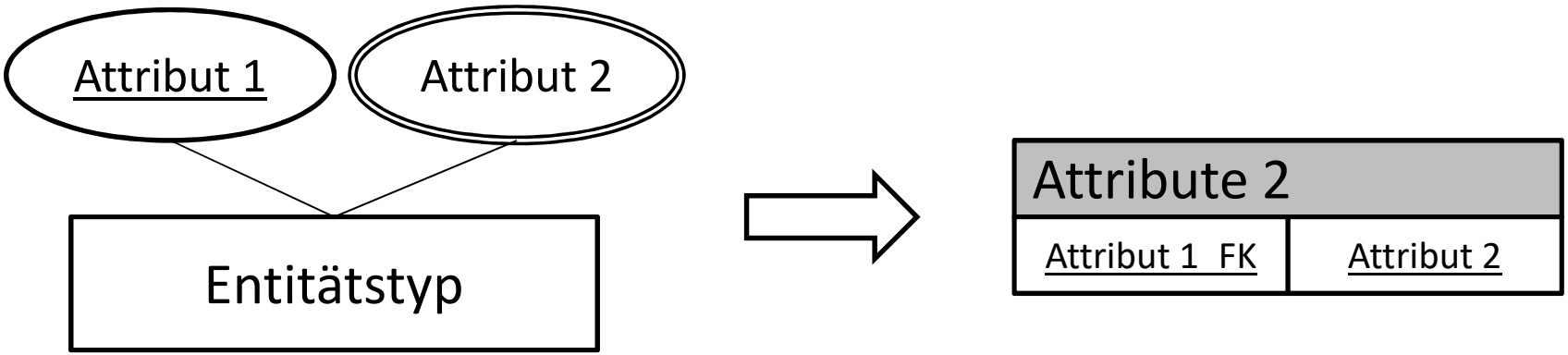
Schritt 5: M:N Beziehungen



Schritt 5: Beziehungstypen mit mehr als zwei beteiligten Entitätstypen und ausschließlich „N“-Kardinalitäten



Schritt 6: Mehrwertige Attribute



2.3 Relationale DB Sprachen

- In diesem Abschnitt betrachten wir verschiedene Datenbanksprachen, mit denen man die Daten in der Datenbank bearbeitet
- Man unterscheidet
  - Zur Bearbeitung der Schemata (Meta Daten) die DML
  - Für die Inhalte die DDL
- Dabei unterscheidet man
  - Prozedurale Anfragesprachen
    - Hier wird festgelegt, wie die Informationen zu ermitteln sind
    - Algebraische Sprachen

      Operationen arbeiten auf gegebenen Relationen und ermitteln wiederum neue Relationen
  - Deskriptive Anfragesprachen
    - Hier wird festgelegt, was ermittelt werden soll
    - Pradikatskalkülbasierte Sprachen (kalkülbasierte Sprachen)

Charakterisierung

- Man unterscheidet wie folgt bei den Sprachen:

Algebraische Sprachen	Prädikatskalkülsprachen	
⇨Verarbeite durch Operationen Relationen zu neuen Relationen	⇨Definiere Relationen durch Formeln	
	Tupelkalkül	Bereichskalkül
	⇨ein Tupel ist atomar	⇨Attribute sind atomar

Relationale Algebra

- Algebra
  - Nichtleere Menge mit abgeschlossenen Operationen
  - Somit erzeugt jede Operation wieder ein Element der betrachteten Menge
  - In unserem Fall besteht die Menge aus Relationen und Schemata
  - Diese werden wieder in neue Relationen und Schemata durch die gegebenen Operationen überführt

- Wir werden im Folgenden zunächst die **Grundoperationen der relationalen Algebra** betrachten

Schemata

2.3.1 **Definition**

Sei R beliebige endliche Relation und ihr Schema „Schema(R)“ gegeben. Dann definieren wir für den Wertebereich der Relation in der relationalen Algebra

$$\Re_R = R \cup \{Schema(R)\}$$

Damit ist jede Operation der Relationalen Algebra in einer der beiden Formen darstellbar:

(i) $\omega : \Re_R \times \Re_S \rightarrow \Re_T$

Beliebige endliche Relationen

(ii) $\omega : \Re_R \rightarrow \Re_S$

Operanden / Operationen

Dabei ist zu beachten:

- Jede Operand in einer solchen Operation ist
  - entweder eine konstante Relation

$$\{(3, Heinz, \ldots)\ ,\ (\ldots)\ ,\ \ldots\ ,\ (\ldots)\}$$

  - oder eine Variable, die wiederum für eine Relation steht

Hierbei sind allerdings Einschränkungen zu beachten. So gilt:

2.3.2 **Einschränkungen**

Es ist im Allgemeinen nicht möglich, die Komplementbildung auf einer Relation R zu erlauben, da das Ergebnis nicht mehr endlich sein muss

Problem der Komplementbildung

- Bilde neue Relation aus einer bestehenden Relation R(Name, Alter)
- Hierbei sollen alle Tupel gebildet werden, die nicht Element der aktuellen Relation sind
- Somit werden unter Umständen unendlich viele neue Einträge generiert, was zu keiner Terminierung der Operation führen kann
- Derartige Operationen sind daher auszuschließen

Arten von Operationen

- Unterschiedliche **Operationen**:
  - (1) **Grundoperation** (GO)
    - Bestimmt die Mächtigkeit der Relationalen Algebra
  - (2) **Abgeleitete Operationen** (AO)
    - Stellt Abkürzungen für Befehlsfolgen von GO bereit
- **Definitionen der Grundoperationen**

  Bei den Definitionen der Grundoperationen:

  - **R, S** stehen im Folgenden für **beliebige endliche Relationen** wobei gilt
    - **R** besitzt das **Schema** $\left(A_1 : D_1, \ \ldots \ , A_n : D_n\right)$
    - **S** besitzt das **Schema** $\left(B_1 : D'_1 \ , \ \ldots \ , B_m : D'_m\right)$

Grundoperation Umbenennung

2.3.3 **Definition**

(0. Grundoperation (nur für Schematabenutzung mit Attributname) Umbenennung)

Die Umbenennung $C = \delta_{A_i \to B}(R)$

wobei $B \notin \{A_1, ..., A_n\}$ $(i \in \{1, ..., n\})$ mit $C = R$ und

$Schema(C) = (A_1 : D_1, \ldots, A_{i-1} : D_{i-1}, B : D_i, A_{i+1} : D_{i+1}, \ldots, A_n : D_n)$

führt zu einer neuen Bezeichnung des Attributes A_i innerhalb der Relation R

Beispiel: Grundoperation Umbenennung

$$C = \delta_{A_1 \to B}(R)$$

R		
A_1	A_2	A_3
1	A	0
4	D	1
…	…	…
100	XY	1

R		
B	A_2	A_3
1	A	0
4	D	1
…	…	…
100	XY	1

Sinn der Operation Umbenennung

- Umbenennungen dienen der eindeutigen Identifizierungen von Attributen innerhalb mehrerer Relationen
- So ist zu beachten, dass unterschiedliche Relationen, die mit Hilfe einer Operation der relationalen Algebra miteinander verknüpft werden, identische Namen aufweisen können
- Zudem kann die Umbenennung im Hinblick auf die Ergebnisfolge der Operation sinnvoll sein

Grundoperation Vereinigung

2.3.4 **Definition**

(1. Grundoperation Vereinigung)

Eine Vereinigung ist $C := S \cup R$,

wobei $Schema(S) = Schema(R)$ gelten muss.

Die Vereinigungsrelation ergibt sich in diesem Fall

aus: $C := \{t \mid t \in S \text{ oder } t \in R\}$.

Zudem gilt: $Schema(C) = Schema(R)\left(= Schema(S)\right)$

Beispiel: Grundoperation Vereinigung

$$C = R \cup S$$

R	
A_1	A_2
1	A
4	D

S	
A_1	A_2
2	H
7	E
4	D

C	
A_1	A_2
1	A
2	H
4	D
7	E

Grundoperation Differenz

2.3.5 **Definition**

(2. Grundoperation: Differenz)

Eine Differenz ist $C := R - S$

wobei $Schema(R) = Schema(S)$ gelten muss.

Damit ergibt sich die resultierende Relation C als:

$C := \{t \mid t \notin S \text{ und } t \in R\}$. Zudem gilt: $Schema(C) = Schema(R)$

Beispiel: Grundoperation Differenz

$$C = R - S$$

R	
A_1	A_2
1	A
4	D

S	
A_1	A_2
2	H
7	E
4	D

C	
A_1	A_2
1	A

Grundoperation Kartesisches Produkt

2.3.6 **Definition**

(3. Grundoperation: Kartesisches Produkt)

Ein Kartesisches Produkt ist $C := R \times S$,

wobei $\{A_1 \ldots A_n\} \cap \{B_1 \ldots B_m\} = \varnothing$ gelten muss. Zudem

gilt: $C := \left\{ (r_1, \ldots, r_n, t_1, \ldots, t_m) \middle| (r_1 \ldots r_n) \in R \text{ und } (t_1 \ldots t_m) \in S \right\}$

mit $Schema(C) = (A_1, \ldots, A_n, B_1, \ldots, B_m)$

Beispiel: Grundoperation Kartesisches Produkt

$$C = R \times S$$

R	
A_1	A_2
1	A
4	D

S	
A_3	A_4
-	null
+	1.0
+	1.7

C			
A_1	A_2	A_3	A_4
1	A	-	null
1	A	+	1.0
1	A	+	1.7
4	D	-	null
4	D	+	1.0
4	D	+	1.7

Grundoperation Projektion

2.3.7 **Definition**

(4. Grundoperation: Projektion)

Eine Projektion ist $C = \pi_{\bar{\Delta}}(R)$

wobei gilt: $\bar{\Delta} = \left(A_{i_1}, \ldots, A_{i_k}\right)$ mit $\left\{A_{i_1}, \ldots, A_{i_k}\right\} \subseteq \left\{A_1, \ldots, A_n\right\}$.

Für die resultierende Relation C gilt: $C := \left\{\pi_{i_1 \ldots i_k}(r) \middle| r \in R\right\} =$

$$\left\{t\left(A_{i_1} : D_{i_1}, \ldots, A_{i_k} : D_{i_k}\right) \middle| \exists s \in R : t\left[A_{i_1}\right] = s\left[A_{i_1}\right] \wedge \ldots \wedge t\left[A_{i_k}\right] = s\left[A_{i_k}\right]\right\}$$

$\forall r = \left(r_1, \ldots, r_n\right) \in R$ schreibt man vereinfachend :

$$\pi_{i_1 \ldots i_k}(r) = \left(r_{i_1}, \ldots, r_{i_k}\right) \text{ mit } Schema(C) = \left(A_{i_1}, \ldots, A_{i_k}\right)$$

Beispiel: Grundoperation Projektion

$$C = \pi_{A_1, A_3}(R)$$

R		
A_1	A_2	A_3
1	A	0
4	D	1
4	E	1
…	…	…
100	XY	1

R	
A_1	A_3
1	0
4	1
…	…
100	1

Grundoperation Selektion

2.3.8 **Definition**

(5. Grundoperation: Selektion)

Eine Selektion ist $C := \sigma_e(R)$ (Beachte: σ ist hier keine Ausprägung).

Zudem ist e eine atomare Formel. Das heißt:

$e = Term \otimes Term$, mit: $\otimes$ Vergleichsoperation auf dem Datentyp des Terms

Ein Term wird rekursiv gebildet:

1. $Term$ ist Konstante oder Attribut eines Tupels aus R

2. $Term = Term \{+, -, *, /\} Term$

Somit gilt für die sich ergebende Relation C:

$C = \{ r \in R \mid r = (r_1 \ldots r_n)$ und die Formel e ist durch Einsetzen von r_i für A_i richtig$\}$. Weiter gilt: $Schema(C) = Schema(R)$

Beispiel: Grundoperation Selektion

$$C = \sigma_{A_1=4}(R)$$

R		
A_1	A_2	A_3
1	A	0
4	D	1
4	E	3
…	…	…
100	XY	1

R		
A_1	A_2	A_3
4	D	1
4	E	3
…	…	…
4	Z	2

Abgeleitete Operationen

- Im Folgenden werden einige komplexere Operationen eingefügt, die wesentlich mächtigere Berechnungen auf der Basis bestehender Relationen durchführen können
- Es wird anschließend gezeigt, dass sich alle diese Operationen durch die dargestellten Grundoperationen simulieren lassen

Abgeleitete Operation Durchschnitt

2.3.9 **Definition**

(1. Abgeleitete Operation: Durchschnitt)

Der Durchschnitt ist $C := R \cap S$ wobei

$Schema(R) = Schema(S)$ gelten muss, mit

$C = \{t | t \in R \text{ und } t \in S\}$ und $Schema(C) = Schema(R)$

Abgeleitete Operation Verallgemeinerte Selektion

2.3.10 **Definition**

(2. Abgeleitete Operation Verallgemeinerte Selektion)

Eine verallgemeinerte Selektion ist $C := \sigma_{\varphi}(R)$

wobei φ eine Formel nach der folgenden rekursiven Definition ist.

Hierbei gilt: Atomare Formeln sind Formeln und

$Formel \otimes Formel$, mit $\otimes \in \{\vee, \wedge\}$ sowie $\neg Formel$ sind Formeln.

Dann gilt für die resultierende Relation C:

$C = \{r \mid r \in R$ und $r = (r_1 \ldots r_n)$ macht φ zu einer richtigen Aussage

durch Einsetzen von r_i für $A_i\}$ und $Schema(C) = Schema(R)$

Abgeleitete Operation Verbund

2.3.11 **Definition**

(3. Abgeleitete Operation: Verbund (Join))

Ein Verbund ist $C := R \underset{A_i \theta B_j}{*} S$, wobei $\theta \in \{=, >, <, \geq, \leq, \neq\}$ mit

$$C = \left\{ \left(r_1, \ldots, r_n, s_1, \ldots, s_m\right) \middle| \left(r_1, \ldots, r_n\right) \in R \text{ und } \left(s_1, \ldots, s_m\right) \in S \right.$$

$$\left. \text{und es gilt: } r_i \theta s_j \right\}$$

Zudem muss gelten: $\{A_1 \ldots A_n\} \cap \{B_1 \ldots B_m\} = \varnothing$, mit

$Schema(C) = (A_1, \ldots, A_n, B_1, \ldots, B_m)$.

Abgeleitete Operation Natürlicher Verbund

2.3.12 **Definition**

(4. Abgeleitete Operation: Natürlicher Verbund (Natural Join))

Ein natürlicher Verbund ist $C := R * S$. Zunächst sei:

$T_{R*S} := dom(A_1) \times \ldots \times dom(A_n) \times dom(B_{k_1}) \times \ldots \times dom(B_{k_e})$, wobei

$\{B_{k_1}, \ldots, B_{k_e}\} := \{B_1, \ldots, B_m\} - \{A_1, \ldots, A_n\}$ und $\forall i \in \{1, \ldots, (e-1)\} : \exists t, t' \in \{1, ..., m\} :$

$(B_{k_i} = B_t \wedge B_{k_{i+1}} = B_{t'}) \Rightarrow t < t'$. Man sieht: Doppelte Attribute verschwinden aber die gegebene Reihenfolge bleibt erhalten. Damit gilt für die resultierende Relation C:

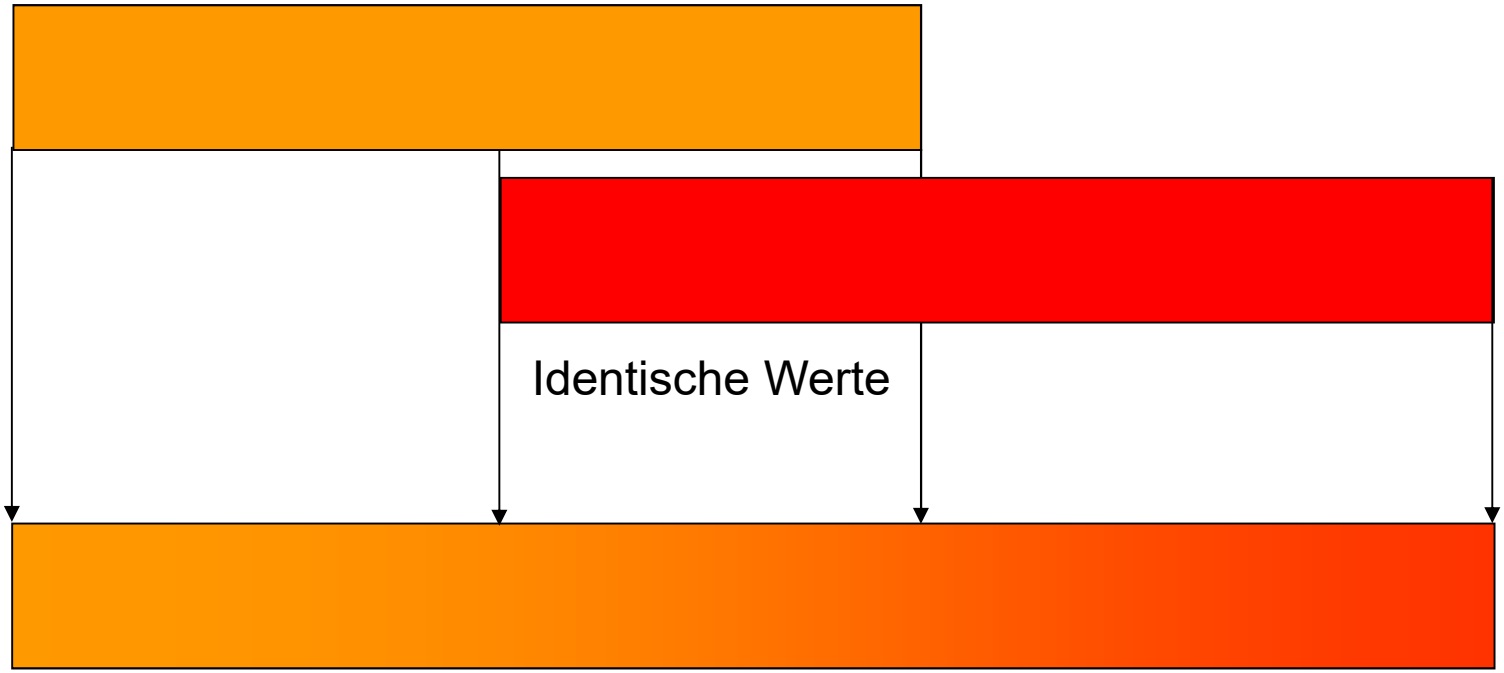
$$C := \{(r_1, \ldots, r_n, s_1, \ldots, s_e) \mid (r_1, \ldots, r_n, s_1, \ldots, s_e) \in T_{R*S} \text{ mit } (r_1, \ldots, r_n) \in R \text{ und } \exists (w_1, \ldots, w_m) \in S :$$

$$\forall i \in \{1, ..., m\} : (\exists t \in \{1, \ldots, e\} : B_i = B_{k_t} \Rightarrow w_i = s_t) \wedge (\exists s \in \{1, \ldots, n\} : B_i = A_s \Rightarrow w_i = r_s)\}$$

Zudem gilt: $Schema(C) = (A_1, \ldots, A_n, B_{k_1}, \ldots, B_{k_e})$.

Illustration – Natürlicher Verbund

- Vereinigung der Tupel



Beispiel – Natürlicher Verbund

- Wir betrachten das folgende Beispiel

Relation U

A	B	C
1	2	3
6	7	8
9	7	8

Relation V

B	C	D
2	3	4
2	3	5
7	8	10

Ergebnis – Natürlicher Verbund

- Damit ergibt sich

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

Abgeleitete Operation Division

2.3.13 **Definition**

(5. Abgeleitete Operation: Division)

Die Division ist $C := R \div S$ wobei $\{B_1 \dots B_m\} \subseteq \{A_1 \dots A_n\}$ und $S \neq \varnothing$ gelten muss.

Dann sei $T_{R \div S} := dom(A_{k_1}) \times \dots \times dom(A_{k_e})$, wobei $\{A_{k_1} \dots A_{k_e}\} = \{A_1 \dots A_n\} - \{B_1 \dots B_m\}$ gilt.

Zudem gilt: $\forall i \in \{1,\dots,(e-1)\} : \exists t, t' \in \{1,...,n\} : (A_{k_i} = A_t \text{ und } A_{k_{i+1}} = A_{t'}) \Rightarrow t < t'$.

(Damit bleibt wiederum die Reihenfolge der Attribute in der Relation R erhalten)

Damit ergibt sich die resultierende Relation C durch:

$$C = \{(r_1 \dots r_e) \mid (r_1,\dots,r_e) \in T_{R \div S} \wedge \forall \bar{b} = (b_1 \dots b_m) \in S : \exists (w_1,\dots,w_n) \in R :$$

$$\forall i \in \{1,2,\dots,n\} : (\exists t \in \{1,\dots,e\} : A_i = A_{k_t} \Rightarrow w_i = r_t) \wedge (\exists t \in \{1,\dots,m\} : A_i = B_t \Rightarrow w_i = b_t)\}$$

Somit ist die sich ergebende Relation C die größte mögliche Relation mit $C \times S \subseteq R$.

Auch gilt $Schema(C) = (A_{k_1},\dots,A_{k_e})$

Division – Konsequenzen

- Jedes Element der Divisionsrelation R÷S besitzt lediglich die Attribute die in R aber nicht in S sind
- Dabei bleibt die Reihenfolge der Attribute in R erhalten
- Zudem muss für jedes Element r der Divisionsrelation R÷S gelten, dass es für alle Elemente s der Relation S ein Element w der Relation R existiert, das
  - bezüglich der gemeinsamen Attribute (das sind die Attribute von S) identisch mit s ist und
  - bezüglich der anderen Attribute identisch mit r ist
- Kombiniert man ein Element der Divisionsrelation R÷S mit den Elementen der Relation S (im Rahmen der Operation „Kartesisches Produkt“) entsteht immer ein Element der Relation R

Simulation abgeleiteter Operationen

- Im Folgenden zeigen wir, dass sich die abgeleiteten Operationen durch Folgen der Grundoperationen simulieren lassen
- Dies kommt in dem folgenden Satz zum Ausdruck

2.3.14 **Satz**

Alle abgeleiteten Operationen sind durch Folgen von Grundoperationen der relationalen Algebra simulierbar

Beweis des Satzes 2.3.14 – Teil 1

(2.3.9) ist simulierbar, da $C = R \cap S = R - \left(R - S\right)$

(2.3.10) $C = \sigma_{\varphi}\left(R\right)$

Induktion über Anzahl n der $\otimes \in \left\{\wedge, \vee\right\}$-Operatoren in der Formel φ

$n = 0 \Rightarrow$ atomare Formel $\Rightarrow$ durch (2.3.1.8) direkt simulierbar.

$n > 0 \Rightarrow$ Voraussetzung: Die Behauptung gilt für alle Formeln mit bis zu n-1 Operatoren. Wir betrachten eine Formel mit n Operatoren

$\text{Fall}\left\langle \text{i} \right\rangle : \varphi = \psi \wedge \pi \Rightarrow C = \sigma_{\psi}\left(R\right) - \left(\sigma_{\psi}\left(R\right) - \sigma_{\pi}\left(R\right)\right)$

$\text{Fall}\left\langle \text{ii} \right\rangle : \varphi = \psi \vee \pi \Rightarrow C = \sigma_{\psi}\left(R\right) \cup \sigma_{\pi}\left(R\right)$

$\text{Fall}\left\langle \text{iii} \right\rangle : \varphi = \neg \psi \Rightarrow C = R - \sigma_{\psi}\left(R\right)$

Da $\sigma_{\psi}\left(R\right)$ und $\sigma_{\pi}\left(R\right)$ nach Induktionsvoraussetzung simulierbar sind folgt die Behauptung

Beweis des Satzes 2.3.14 – Teil 2

(2.3.11)

$$C = R \underset{A_i \theta B_j}{*} S = \sigma_{A_i \theta B_j}\left(R \times S\right)$$

(2.3.12)

Sei o.B.d.A. $\left\{B_1,...,B_k\right\} = \left\{A_1,...,A_n\right\} \cap \left\{B_1,...,B_m\right\}$. Damit gilt:

$$C = \pi_{A_1,...,A_n,B_{k+1},...,B_m}\left(\sigma_{B_1=C_1}\left(.... \sigma_{B_{k-1}=C_{k-1}}\left(\sigma_{B_k=C_k}\left(R \times \delta_{B_1 \to C_1,...,B_k \to C_k}\left(S\right)\right)\right)\right)\right)$$

(2.3.13)

$C := R \div S$ Voraussetzung ist hierfür: $\left\{B_1,...,B_m\right\} \subseteq \left\{A_1,...,A_n\right\}$.

O.B.d.A. sei $\left\{A_{n-m+1},...,A_n\right\} = \left\{B_1,...,B_m\right\}$

$$C = \pi_{A_1,...,A_{n-m}}\left(R\right) - \left(\pi_{A_1,...,A_{n-m}}\left(\left(\pi_{A_1,...,A_{n-m}}\left(R\right) \times S\right) - R\right)\right)$$

2.3.15 Weitere Beispiele

- (i) Division:

	A	B	C	D
R=	1	2	3	4
	1	2	5	6
	7	8	3	4
	9	10	11	12

	C	D
S=	3	4
	5	6

	A	B
R÷S=	1	2

Division…was ist das denn nun wieder?

Ein weiteres Beispiel für eine Division

- Wir betrachten folgende Relationen

Party

PartyID	Veranstalter	Datum	Motto
1	JustinB	27.3.2016	Winfor is over now
…	…	…	…

Gast

Besucher	PartyID_FK
Anna	1
…	…

Allergie

Allergiker	Allergen
Anna	Haselnuss
…	…

- Die Gäste auf Susannes legendärer und einmalig von ihr durchgeführter „Bad Taste“ Party bilden eine eingeschworene Clique. Auf welchen Partys (PartyID, Veranstalter, Datum, Motto) war diese Clique vollständig vertreten?
- Um an diese Information zu gelangen, führen wir eine Division durch (D = R ÷ S)
- Hierfür müssen wir zunächst die Relation R mit den benötigten Daten bilden

$$R = \pi_{PartyID,Veranstalter,Datum,Motto,Besucher}\left(\sigma_{PartyID=PartyID_FK}\left(Party \times Gast\right)\right)$$

- und die Gäste der legendären Party ermitteln

$$S = \pi_{Besucher}\left(\sigma_{Motto="Bad\ Taste"}\left(\sigma_{Veranstalter=Susanne}\left(\sigma_{PartyID=PartyID_FK}\left(Party \times Gast\right)\right)\right)\right)$$

Konsequenz

- Damit ergibt sich durch die Operation

$$D = R \div S$$

das gewünschte Ergebnis

- Die Division verlangt ja gerade, dass in R mindestens alle Kombinationen von D und S sind, also jeder Gast aus S (die Teilnehmer der legendären „Bad Taste“-Party von Susanne (die ja eindeutig ist)) taucht auf der Teilnehmerliste der Partys auf, die in D aufgeführt sind.
- Gäbe es einen Gast aus S, der nicht auf der betrachteten Party in D war, kann diese Party nicht in D sein, da dieses Tupel sich nicht in R findet. R ist aber eine Obermenge des kartesischen Produktes von D und S

$$D \times S \subseteq R$$

Weitere Beispiele

- (ii) Natürlicher Verbund

	A	B	C
R=	1	2	3
	5	2	3
	2	2	6
	3	1	4

	B	C	D
S=	2	3	4
	2	3	6
	1	4	2

	A	B	C	D
R*S=	1	2	3	4
	1	2	3	6
	5	2	3	4
	5	2	3	6
	3	1	4	2

Entwicklung von typischen Anfragen

- Im Folgenden betrachten wir ein kleines Beispiel, um die Anwendung der Operationen der relationalen Algebra zu erlernen
- Dazu betrachten wir die folgende Ausprägung einer relationalen Datenbank über die gegenseitigen Besuche, Beziehungen und Wohnverhältnisse von Studierenden

Relation PÄRCHEN	
MName	**WName**
Markus	Birgit
Frank	Simone
Rainer	Nicole
Andreas	Katja

Relation WOHNUNG		
Vermieter	**Hauptmieter**	**Miete**
Schulze	Andreas	650
Schulze	Nicole	1100
Müller	Tanja	450

Weitere Relationen sind…

Relation BESUCHE		
Gastgeber	**Gast**	**Datum**
Tanja	Markus	30.10.2007
Tanja	Rainer	30.10.2007
Frank	Simone	30.10.2007
Rainer	Birgit	04.11.2007
Nicole	Wolfgang	04.11.2007
Nicole	Simone	04.11.2007
Nicole	Andreas	04.11.2007
Rainer	Nicole	07.11.2007
Katja	Birgit	08.11.2007
Frank	Simone	08.11.2007

Relation WOHNGEMEINSCHAFT	
Hauptmieter	**Untermieter**
Andreas	Simone
Nicole	Katja
Nicole	Markus
Tanja	Birgit
Nicole	Frank
Andreas	Wolfgang
Nicole	Rainer

Mögliche Anfragen

- Formulieren Sie die folgenden vier Anfragen in der Relationalen Algebra. Bitte verwenden Sie hierbei **ausschließlich die Grundoperationen der Relationalen Algebra**. Geben Sie auch jeweils die sich ergebenden Relationen an.

  1. An welchen Tagen hat Rainer Besuch bekommen?
  2. Wer ist Untermieter und hat Besuch bekommen?
  3. Wessen Freund wohnt zur Untermiete bei Nicole?
  4. Welche Personen sind Untermieter in einer Wohnung und haben am 04.11.2007 jemanden besucht?

Anfrage 1

- An welchen Tagen hat Rainer Besuch bekommen?

$$\pi_{Datum}\left(\sigma_{Gastgeber="Rainer"}\left(BESUCHE\right)\right)$$

- Ergebnisrelation

Ergebnisrelation
Datum
04.11.2007
07.11.2007

Anfrage 2

- Wer ist Untermieter und hat Besuch bekommen?

$$\pi_{Gastgeber}\left(\sigma_{Gastgeber=Untermieter}\left(BESUCHE \times WOHNGEMEINSCHAFT\right)\right)$$

- Ergebnisrelation

Ergebnisrelation
Gastgeber
Frank
Rainer
Katja

Anfrage 3

- Wessen Freund wohnt zur Untermiete bei Nicole?

$$\pi_{WName}\left(\sigma_{Hauptmieter="Nicole"}\left(\sigma_{MName=Untermieter}\left(PÄRCHEN \times WOHNGEMEINSCHAFT\right)\right)\right)$$

- Ergebnisrelation

Ergebnisrelation
WName
Birgit
Simone
Nicole

Anfrage 4

- Welche Personen sind Untermieter in einer Wohnung und haben am 04.11.2007 jemanden besucht?

$$\delta_{Gast \rightarrow Person}\left(\pi_{Gast}\left(\sigma_{Datum="04.11.2007"}\left(\sigma_{Gast=Untermieter}\left(BESUCHE \times WOHNGEMEINSCHAFT\right)\right)\right)\right)$$

- Ergebnisrelation

Ergebnisrelation
Person
Birgit
Wolfgang
Simone

2.4 Einführung in die Design Theorie

- Im Folgenden wollen wir verstehen, was ein „gutes“ Datenbank Design auszeichnet
- Wann sagen wir, dass ein Datenbankschema leistungsfähig im Hinblick auf unsere Anforderungen ist
- Dazu müssen wir uns erst einmal unsere Anforderungen klar machen
- Dies geschieht nun auf den nächsten Folien

2.4.1 Semantik der Attribute

- Unter der Semantik versteht man allgemein die „Interpretation“ oder „Bedeutung“ eines Attributes
- Die Qualität eines Schemas hängt auch davon ab, wie einfach sich die Bedeutung eines Attributes ableiten lässt

Person	Straße	ID	NName	HN	PLZ	Name

Person	Straße	Personal-ausweis-kennziffer	Nach-name	Haus-nummer	PLZ	Vorname

- Problematisch ist in diesem Zusammenhang insbesondere die Bildung von Joins bzw. kartesischen Produkten,
  - da hier sehr unübersichtliche Relationen entstehen,
  - und Attribute häufig nicht auf eine Gemeinsamverwendung vorbereitet sind

Join Problematik

- Zwei Relationen

Stadion	Straße	ID	Name	Haus-nummer	PLZ	Kapazität

Künstler	KStraße	Nach-name	Geburtsjahr	KName	KHaus-nummer	KPLZ	Vor-name

- Die Ergebnisrelation
  - Semantik sehr unklar, Mehrdeutigkeit ist festzustellen
  - Wofür steht Name? Wofür KName?

Stadion x Künstler	Straße	ID	Name	Hausnummer	PLZ	Kapazität
KStraße	Nach-name	Geburts-jahr	KName	KHaus-nummer	KPLZ	Vorname

Einfache Folgerung

- Je leichter man die Bedeutung von verwendeten Attributen in einer Relation erkennen kann, desto besser ist das entworfene Schema

2.4.2 Normalformen

- In der Literatur finden sich einige Normalformen, die einige formale Anforderungen an das Design von Schemata umsetzen
- Diese bauen aufeinander auf und werden im Folgenden nacheinander betrachtet

2.4.2.1 Erste Normalform

- Ein einfaches Schema ($R(A_1,\ldots,A_n),F$) ist in erster Normalform falls alle Attribute über Mengen von einfachen atomaren Werten definiert sind
- Grundsätzlich ist die erste Normalform Teil der allgemeinen Definition von Relationen
- Sie legt lediglich fest, dass Attribute nicht
  - mehrwertig oder
  - zusammengesetzt sind
- Beispiel
  - Department(Name, Nummer, Dep_Code, {Standorte})
  - Es gilt: Nummer→Name; Nummer→Dep_Code; Nummer→{Standorte}

Erste Normalform – Beispiel

- Man sieht, dass die Relation nicht in erster Normalform ist
  - Mehrwertiges Attribut {Standorte}
  - Dieses widerspricht der Definition
- Umwandlung wird notwendig
  - Department(Name, Nummer, Dep_Code, Standort)
  - Damit nur noch atomare Werte
  - Erste Normalform erreicht

2.4.2.2 Funktionale Abhängigkeiten

2.4.2.2.1 **Definition**

Sei $R(A_1,\ldots,A_n)$ ein Relationsschema und seien zudem X und Y Teilmengen aus $\{A_1,\ldots,A_n\}$.

Dann sagt man dass $X \rightarrow Y$ (d.h. Y hängt funktional von X ab), wenn die folgenden Eigenschaften gelten:

Falls für zwei Tupel r und s in $\mu(R)$ $r[X]=s[X]$ gilt, dann gilt auch $r[Y]=s[Y]$

2.4.2.2.2 **Definition**

Ein Schema R beinhaltet neben den Attributen auch die bestehenden funktionalen Abhängigkeiten F. Falls R die funktionalen Abhängigkeiten F erfüllt, schreibt man einfach R»F

Triviale funktionale Abhängigkeiten

2.4.2.2.3 **Definition**

Sei $R(A_1,\ldots,A_n)$ ein Relationsschema und X→Y eine funktionale Abhängigkeit laut Definition 2.4.2.2.1. Dann heißt X→Y trivial genau dann wenn $Y \subseteq X$ gilt.

Der Schlüssel einer Relation…

- …ist ein erstes einfaches Beispiel einer funktionalen Abhängigkeit
- Von einem Schlüssel hängen alle anderen Attributsausprägungen in einem Tupel ab

Person	Straße	Personal-ausweis-kennziffer	Nach-name	Haus-nummer	PLZ	Vorname

- Man sieht:
  - Jede Personalausweiskennziffer ist eindeutig
  - Damit gibt es niemals zwei Tupel mit unterschiedlichen Einträgen in anderen Attributen bei gleicher Personalausweiskennziffer

Was ist das Problem dabei?

- Wenn wir eine Funktionale Abhängigkeit X→Y in einer Relation vorliegen haben, dann besteht die Gefahr von Redundanzen
- Kleines Beispiel

Spieler	SName	SAlter	VName	VTrainer	VAnschrift	Vgegründet

Funktionale Abhängigkeiten

  - SName → VName, SAlter, VTrainer, VAnschrift, Vgegründet
  - VName → VTrainer, VAnschrift, Vgegründet

- Damit führt jede Eintragung eines Spielers desselben Vereins zu Redundanz und es ist auf eine konsistente Belegung zu achten
- So stellt bereits der zweite Teil eine Aufspaltung in eine neue Relation in Aussicht

Was ist eigentlich F?

- Die Menge der Funktionalen Abhängigkeiten F beschreibt das Wissen, das man über die Zusammenhänge der Attributswerte in einer Relation hat
- Es zu entwickeln ist nicht nur aus informationstechnischen Gründen wichtig
- Wissenserwerb, -erhalt und intersubjektiv nachvollziehbare Aufbereitung ist zunehmend entscheidend für den Geschäftserfolg
- Daher ist eine Pflege des Wissens von besonderer Bedeutung

Inferenzregeln zur Ableitung neuen Wissens

Gegeben: (R(A),F), A: Attributsmenge der Relation R, F: Menge der gegebenen funktionalen Abhängigkeiten

Dann dienen die folgenden Inferenzregeln zur Ableitung neuen Wissens:

- **Reflexivität**: Falls X‘ eine Teilmenge von X (Teilmenge aus A) ist dann lässt sich X→X‘ in F ableiten
- **Erweiterung**: Falls X→Y in F und Z eine Teilmenge aus A ist dann lässt sich auch XZ→YZ ableiten
- **Transitivität**: Falls X→Y und Y→Z in F dann lässt sich auch X→Z ableiten

Diese Regeln werden auch **Armstrong Axiome** genannt

Korrektheit und Vollständigkeit

- Man kann zeigen, dass die Armstrong Axiome
  - korrekt und
  - vollständig sind
- Formal heißt das, dass falls sich aus einer Menge von Attributen X die Attributsmenge Y aufgrund der Regeln in F logisch ableiten lässt, dies auch durch Anwendung der Armstrong Axiome gelingt
- Dies bedeutet, dass es kein Wissen (also keine funktionale Abhängigkeit) gibt, das sich nicht durch Anwendung der drei Inferenzregeln ableiten lässt

Abschluss

- Für eine Menge von Attributen X bezeichnen wir mit dem Abschluss X^+ die Attributsmenge, die sich durch Anwendung der Armstrong Axiome mit Hilfe der funktionalen Abhängigkeiten in F aus X ableiten lässt. Dies heißt formal:

$$X^+ = \left\{ Y \in A \,\middle|\, \begin{array}{l} X \rightarrow Y \text{ ist allein durch Anwendung der} \\ \text{Armstrong Axiome in F ableitbar, d.h. man} \\ \text{kann Y aus X in F ableiten} \end{array} \right\}$$

Beispiel

- Film(Titel, Jahr, Länge, Genre, Studio, Studio_Adresse)
- Funktionale Abhängigkeiten
  - Titel, Jahr → Studio
  - Studio → Studio_Adresse
- Wir können mit Hilfe der Armstrong Axiome sofort sehen, dass beispielsweise gilt
  - Titel, Jahr → Studio_Adresse (Transitivität)
  - Titel, Jahr, Genre → Studio, Genre (Erweiterung)

Berechnung von X^+

Input: X

$X^+=X$;

Wiederhole

alt_$X^+=X^+$;

Für jedes ($Y \rightarrow Z$) aus F mit Y als Teilmenge von X^+
ersetze X^+ durch $X^+ \cup Z$

Solange bis gilt (alt_$X^+=X^+$)

/* kein neues Wissen mehr erhalten */

Output: X^+

Korrektheit

- In dem Algorithmus werden alle Attribute aus X abgeleitet, die in F aus X logisch folgen
- Da die Armstrong vollständig sind errechnet der Algorithmus also korrekt den Abschluss einer Menge X

Beispiel

- Gegeben sei das Schema

  (Name, Datum, Straße, Treffpunkt, Auto, Aktivität)

  Hiermit sollen Ihre Treffen protokolliert werden (mit wem, der in der Straße Straße wohnt, haben Sie sich wo getroffen und sind mit welchem Auto (oder ohne Auto) hingefahren. Zudem haben Sie u.U. mehrere Aktivitäten durchgeführt)

- Abhängigkeiten
  - Name → Straße
  - Name, Datum → Treffpunkt
  - Name, Datum → Auto
  - Treffpunkt → Datum

Wir erzeugen {Name, Treffpunkt}$^+$

- Wir beginnen {Name, Treffpunkt}
- Im ersten Schritt ist die Regel 1 anwendbar. Wir erhalten also {Name, Treffpunkt, Straße}
- Damit können wir nun die vierte Regel anwenden und erhalten {Name, Treffpunkt, Straße, Datum}
- Jetzt ist auch die Regeln drei anwendbar. Damit ergibt sich {Name, Treffpunkt, Straße, Datum, Auto}
- Nun lassen sich keine weiteren neuen Attribute ableiten und es ergibt sich als Abschluss {Name, Treffpunkt, Straße, Datum, Auto}

2.4.2.3 Zweite Normalform (nach Codd)

Ein einfaches Schema ($R(A_1,\ldots,A_n),F$) ist in zweiter Normalform falls

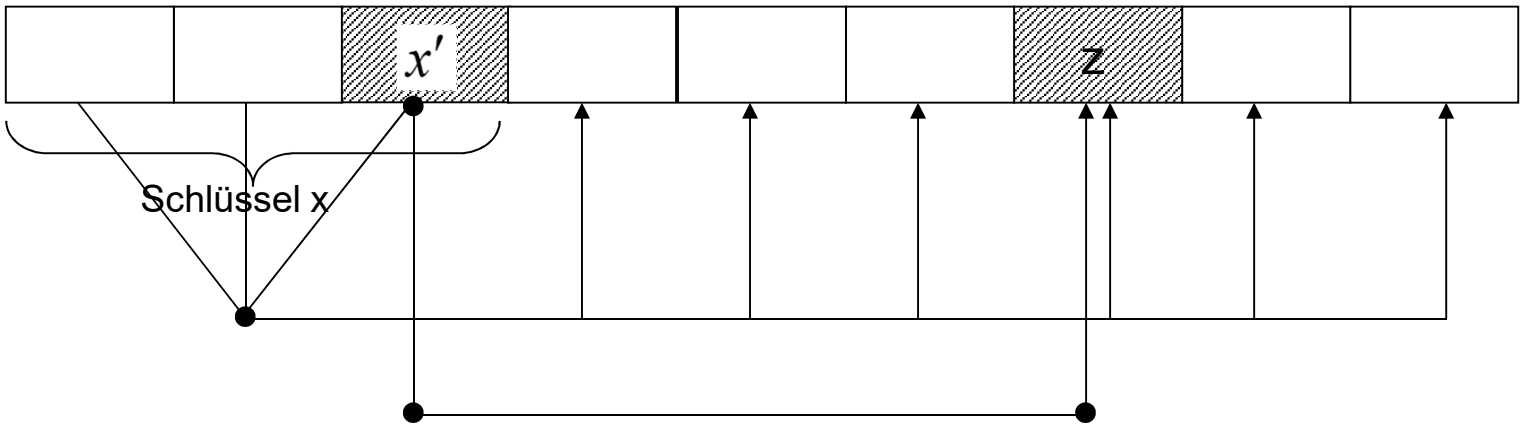
- Das Schema die Auflagen der ersten Normalform erfüllt und
- kein Nichtschlüsselattribut von einem Schlüssel nur partiell abhängt

Sei $SK = \{X_1,\ldots,X_n\}$ die Menge aller Schlüsselkandidaten des Schemas $(R(A_1,\ldots,A_n),F)$ mit $\forall i \in \{1,\ldots,n\} : X_i \subseteq \{A_1,\ldots,A_n\}$. Dann gilt für die zweite Normalform nach Codd:

$$\forall z \in \left(\{A_1,\ldots,A_n\} - \bigcup_{i=1}^{n} X_i \right) : \forall X_i \in SK : \neg\exists Y : (Y \subset X_i \wedge Y \rightarrow z)$$

Zweite Normalform – Illustration

- Man sieht, dass z von x‘ alleine abhängt
- Das heißt, dass alle Tupel mit gleichem x‘ auch ein identischen Eintrag für z haben müssen
- Speicherplatzverschwendung und Redundanz
- Gefahr von möglichen Inkonsistenzen



Test auf Einhaltung der zweiten Normalform

Gegeben: $(R(A_1,\ldots,A_n),F)$ und F hat nur einelementige rechte Seiten

Überprüfe für jede funktionale Abhängigkeit $(X \rightarrow A) \in F$ in der A nicht Teil der Menge X ist ob

1. X *keine* echte Teilmenge eines Schlüssels ist **oder**
2. A ein Schlüsselattribut ist.

Falls (1) **und** (2) für nur eine funktionale Abhängigkeit nicht zutreffen, ist die zweite Normalform verletzt.

Beispiel: Test auf Einhaltung der 2.NF

Spieler(Vorname, Nachname, Geburtsdatum, Beitrittsdatum, Rolle, Mitgliedsbeitrag, Mitgliedsnummer)

Vorname, Nachname → Geburtsdatum	X ist keine echte Teilmenge eines Schlüssels, A ist kein Schlüsselattribut
Vorname, Nachname → Beitrittsdatum	s.o., A ist kein Schlüsselattribut
Vorname, Nachname → Rolle	s.o., A ist kein Schlüsselattribut
Vorname, Nachname → Mitgliedsbeitrag	s.o., A ist kein Schlüsselattribut
Mitgliedsnummer → Geburtsdatum	s.o., A ist kein Schlüsselattribut
Mitgliedsnummer → Beitrittsdatum	s.o., A ist kein Schlüsselattribut
Mitgliedsnummer → Rolle	s.o., A ist kein Schlüsselattribut
Mitgliedsnummer → Mitgliedsbeitrag	s.o., A ist kein Schlüsselattribut
Vorname, Nachname → Mitgliedsnummer	s.o., A ist Schlüsselattribut
Mitgliedsnummer → Vorname	s.o., A ist Schlüsselattribut
Mitgliedsnummer → Nachname	s.o., A ist Schlüsselattribut
Rolle → Mitgliedsbeitrag	s.o., A ist kein Schlüsselattribut

Kein Verstoß gegen die 2.NF

Zweite Normalform – Beispiel

- AUSLEIHE(DokNr, Titel, Autor, Verlag, Deskriptor, MatrNr, Student, Semester, Fachrichtung, Ausleihdatum)
- DokNr → Titel
- DokNr → Verlag
- DokNr → Autor
- MatrNr → Student
- MatrNr → Semester
- MatrNr → Fachrichtung
- Autor → Verlag
- DokNr, MatrNr → Ausleihdatum

Zweite Normalform – Beispiel

- Wir erhalten somit den folgenden Schlüssel

  (DokNr, MatrNr, Deskriptor)
- Die Relation ist offensichtlich nicht in zweiter Normalform, da beispielsweise Titel, Verlag und Autor nur von DokNr abhängen, also nur partiell vom Schlüssel
- Damit müssen bei jeder Ausleihe mit derselben DokNr immer wieder diese Daten angegeben werden
- Hieraus entstehen Redundanz- und Konsistenzprobleme
- Daher empfiehlt sich eine Aufteilung

2.4.2.4 Minimale Überdeckung

- Im Folgenden soll die Menge F der funktionalen Abhängigkeiten umgestaltet werden
- Dazu wird zunächst eine sogenannte „Minimale Überdeckung“ G ermittelt, die folgende Eigenschaften hat
  - Es sollen bezüglich aller Attributsmengen dieselben Abschlüsse erzielbar sein, d.h. G und F sind äquivalent im Hinblick auf die Ableitbarkeit von neuen Attributen aus bekannten. Wir schreiben hierfür kurz: $F^+=G^+$
  - Die Menge der vorliegenden Abhängigkeiten soll eine mögliche Zerlegung erleichtern

2.4.2.4 Minimale Überdeckung

Definition

Seien F und G Mengen funktionaler Abhängigkeiten. Dann ist G eine minimale Überdeckung zu F falls gilt:

- $G^+=F^+$ (identisches Wissen ableitbar), d.h. für alle Attributsmengen X gilt, dass X^+ unter Einsatz von F identisch mit X^+ unter Einsatz von G ist
- Für jedes G‘ das Teilmenge von G ist gilt: Falls $G^+=G'^+$ dann folgt G‘=G (G ist also minimal gewählt)
- Rechte Seiten von G haben Größe 1
- Linke Seiten sind minimal. D.h. falls X→Y aus G und X‘ ist Teilmenge von X dann gilt: Falls man in G (X→Y) durch (X‘→Y) ersetzt und weiterhin $G^+=F^+$ folgt, dann gilt X‘=X

Berechnung einer minimalen Überdeckung

1. Ersetze alle rechten Seiten durch einelementige rechte Seiten, d.h. aus $X \rightarrow A_1,\ldots,A_n$ werden die neuen Abhängigkeiten $X \rightarrow A_1,\ldots, X \rightarrow A_n$
2. Für alle Regeln $X \rightarrow A_i$ aus F und alle B aus X überprüfe ob A_i aus $(X\text{-}B)^+$ gilt (für F)
   - Ja, dann ersetze die Regel durch $(X\text{-}B) \rightarrow A_i$
   - Nein. Lasse die Regel so bestehen
3. Für alle Regeln $X \rightarrow A_i$ aus F überprüfe ob A_i aus X^+ gilt für F / $\{X \rightarrow A_i\}$
   - Ja, dann entferne die Regel (unnötig)
   - Nein. Die Regel muss beibehalten werden

Minimale Überdeckung – Beispiel

- Wir betrachten die folgende Relation
  PERSON(Name, Vorname, Alter, PLZ, Straße, HNr)
  Es gelten die folgenden funktionalen Abhängigkeiten
  - Name, Vorname → Alter, PLZ, Straße, HNr
  - Straße, HNr → Name
  - Straße, HNr → PLZ
  - Name → PLZ
- Wir bestimmen die minimale Überdeckung
  - Name, Vorname → Alter
  - Name, Vorname → PLZ
  - Name, Vorname → Straße
  - Name, Vorname → HNr
  - Straße, HNr → Name
  - Straße, HNr → PLZ
  - Name → PLZ

Minimale Überdeckung – Beispiel

- Die Regel (Straße, HNr → PLZ) lässt sich durch die beiden Regeln Straße, HNr → Name und Name → PLZ ableiten und wird daher gestrichen
- Zudem kann die Regel Name, Vorname → PLZ gestrichen werden
- Damit ergibt sich als minimale Überdeckung
  - Name, Vorname → Alter
  - Name, Vorname → Straße
  - Name, Vorname → HNr
  - Straße, HNr → Name
  - Name → PLZ

Zerlegung eines Schemas

- Ein Schema $R(A_1,\ldots,A_n)$ soll in k Teile (neue Relationen) zerlegt werden
- Eine solche Zerlegung $\{R_1,\ldots,R_k\}$ hat die folgenden Eigenschaften
  - R_i ist eine Teilmenge von $\{A_1,\ldots,A_n\}$
  - $R_1 \cup \ldots \cup R_k = \{A_1,\ldots,A_n\}$ (Kein Verlust von Attributen)
- Eine solche Zerlegung ist verlustlos wenn gilt:

$$\forall r \subseteq \mu(R) : \pi_{R_1}(r) * \ldots * \pi_{R_k}(r) = r$$

- Das heißt, wir haben keinen Verlust von Tupeln, also unserer Daten!
- Es entstehen auch keine neuen Tupel!

2.4.2.5 Dritte Normalform

Ein einfaches relationales Schema $(R(A_1,\ldots,A_n),F)$ ist in dritter Normalform falls

- das Schema die Auflagen der ersten Normalform erfüllt
- und kein Nichtschlüsselattribut (NSA) von einem Schlüsselkandidaten nicht trivial transitiv abhängt, d.h. es existiert kein NSA A_i und Menge von Attributen (von R) Y mit den Eigenschaften:
  - $X \rightarrow Y$, $Y \rightarrow A_i$,
  - wobei X nicht aus Y herleitbar ist und
  - A_i nicht in Y ist.

Dritte Normalform – Interpretation

- Wenn ein NSA von einer Menge von Attributen Y eines Schemas, das in der dritten Normalform ist, nicht trivial abhängt, dann kann die transitive Abhängigkeit von jedem Schlüsselkandidaten nur dadurch aufgehoben werden, dass Y selbst ein Schlüsselkandidat ist (ein formaler Beweis hierfür erfolgt noch im Kapitel 2.4.2.9)
- Formal heißt das:

Sei $SK = \{X_1, ..., X_n\}$ die Menge aller Schlüsselkandidaten des Schemas $(R(A_1, ..., A_n), F)$ mit $\forall i \in \{1, ..., n\} : X_i \subseteq \{A_1, ..., A_n\}$.

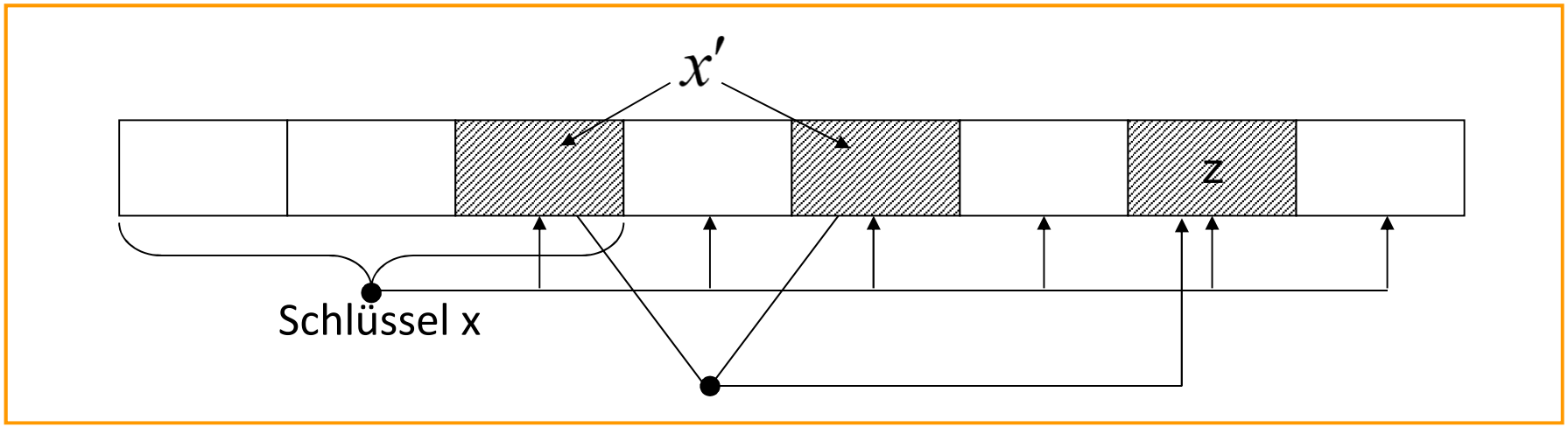
Dann gilt für die dritte Normalform:

$$\forall z \in \left(\{A_1, ..., A_n\} - \bigcup_{i=1}^{n} X_i \right) : \forall Y \subseteq \{A_1, ..., A_n\} : \forall X_i \in SK :$$

$$(X_i \rightarrow Y \wedge Y \rightarrow z) \Rightarrow Y \in \{X_1, ..., X_n\}$$

Dritte Normalform – Illustration

- Für jede Belegung x‘ muss z entsprechend gewählt werden
- Damit wiederum Redundanz und Inkonsistenzgefahr



Test auf Einhaltung der dritten Normalform

Gegeben: $(R(A_1,\ldots,A_n),F)$ und F hat nur einelementige rechte Seiten

Überprüfe für jede funktionale Abhängigkeit $(X \rightarrow A) \in F$ in der A nicht Teil der Menge X ist ob

1. X ein Superschlüssel **oder**
2. A ein Schlüsselattribut *ist*.

Falls (1) **und** (2) für nur eine funktionale Abhängigkeit **nicht** zutreffen, ist die dritte Normalform verletzt.

Relationen mit mehreren Schlüsselkandidaten

Spieler	Vorname	Nachname	Geburts-datum	Mitglieds-nummer	Beitritts-datum	Rolle	Mitglieds-beitrag

- Es gelten folgende funktionale Abhängigkeiten:
  - Vorname, Nachname → Geburtsdatum, Beitrittsdatum, Rolle, Mitgliedsbeitrag
  - Mitgliedsnummer → Geburtsdatum, Beitrittsdatum, Rolle, Mitgliedsbeitrag
  - Vorname, Nachname → Mitgliedsnummer
  - Mitgliedsnummer → Vorname, Nachname
  - Rolle → Mitgliedsbeitrag
- Somit sind sowohl (Vorname, Nachname) als auch (Mitgliedsnummer) Schlüsselkandidaten
- Die Attribute der Relation Spieler bilden zwei disjunkte Teilmengen
- Schlüsselattribute sind: Vorname, Nachname, Mitgliedsnummer
- Nichtschlüsselattribute sind: Geburtsdatum, Beitrittsdatum, Rolle, Mitgliedsbeitrag

Beispiel: Test auf Einhaltung der 3.NF

Vorname, Nachname → Geburtsdatum	X ist Superschlüssel
Vorname, Nachname → Beitrittsdatum	X ist Superschlüssel
Vorname, Nachname → Rolle	X ist Superschlüssel
Vorname, Nachname → Mitgliedsbeitrag	X ist Superschlüssel
Mitgliedsnummer → Geburtsdatum	X ist Superschlüssel
Mitgliedsnummer → Beitrittsdatum	X ist Superschlüssel
Mitgliedsnummer → Rolle	X ist Superschlüssel
Mitgliedsnummer → Mitgliedsbeitrag	X ist Superschlüssel
Vorname, Nachname → Mitgliedsnummer	X ist Superschlüssel A ist Schlüsselattribut
Mitgliedsnummer → Vorname	X ist Superschlüssel A ist Schlüsselattribut
Mitgliedsnummer → Nachname	X ist Superschlüssel A ist Schlüsselattribut
Rolle → Mitgliedsbeitrag	X ist kein Superschlüssel und A ist kein Schlüsselattribut

Verstoß gegen die 3.NF

Dritte Normalform – Schlussfolgerungen

2.4.2.5.1 **Lemma**

Ist ein relationales Schema ($R(A_1,\ldots,A_n),F$) in dritter Normalform dann ist es auch in zweiter Normalform

Beweis:

- Wir nehmen das Gegenteil an. Das heißt, das Schema ist in dritter, aber nicht in zweiter Normalform
- Somit gibt es ein NSA z, das von einem Schlüssel X partiell abhängt
- Es hängt somit von X‘ (eine echte Teilmenge von X) ab
- Damit gilt, dass eine transitive Abhängigkeit von X besteht
- Nämlich gilt nun: $X \rightarrow X' \rightarrow z$
- Dies ist ein Widerspruch zur Annahme der Erfüllung der dritten NF

2.4.2.5.2 Transformationsalgorithmus

1. Bestimme die (eine) minimale Überdeckung G von F.
2. Falls $X \rightarrow Y$ aus G mit $X \cup Y = \{A_1,\ldots,A_n\}$, so ist (R,G) bereits in 3. Normalform, dann: Abbruch

Im anderen Fall:

3. Gruppiere die Funktionalen Abhängigkeiten von G nach gleichen linken Seiten X. Sei für eine Gruppe gleicher linken Seiten die Menge U_i alle in diesen Regeln vorkommenden Attribute (aller linken und rechten Seiten). Wenn noch kein Schema existiert, das alle Attribute aus U_i besitzt, dann definiere ein Teilschema (U_i,F_i), wobei $F_i = \pi_{U_i}(G)$
4. Falls der Gesamtschlüssel in keinem Schema auftaucht, dann füge ein weiteres Schema mit dem Gesamtschlüssel der ursprünglichen Relation hinzu

Umwandlung – Beispiel

- Wir betrachten wieder unsere Bibliothek
- AUSLEIHE(DokNr, Titel, Autor, Verlag, Deskriptor, MatrNr, Student, Semester, Fachrichtung, Ausleihdatum)
- DokNr → Titel
- DokNr → Verlag
- DokNr → Autor
- MatrNr → Student
- MatrNr → Semester
- MatrNr → Fachrichtung
- Autor → Verlag
- DokNr, MatrNr → Ausleihdatum

Umwandlung – Beispiel

1. Minimale Überdeckung finden
   - Dies geht sehr schnell, da nur eine Regel überflüssig ist
   - Wir können direkt DokNr → Verlag entfernen, da es ableitbar ist aus
   - DokNr → Autor und Autor → Verlag
   - Damit ist bereits G als minimale Überdeckung gefunden
     - DokNr → Titel
     - DokNr → Autor
     - MatrNr → Student
     - MatrNr → Semester
     - MatrNr → Fachrichtung
     - Autor → Verlag
     - DokNr, MatrNr → Ausleihdatum

Umwandlung – Beispiel

2. Leider gibt es keine volle Überdeckung durch eine Regel
3. Damit geht es weiter mit Schritt 3. Wir betrachten die Regeln
   - DokNr → Titel
   - DokNr → Autor

   Buch(<u>DokNr</u>, Titel, Autor)
   - MatrNr → Student
   - MatrNr → Semester
   - MatrNr → Fachrichtung

   Student(<u>MatrNr</u>, Student, Semester, Fachrichtung)
   - Autor → Verlag

   Kontrakt(<u>Autor</u>, Verlag)
   - DokNr, MatrNr → Ausleihdatum

   Ausleihe(<u>DokNr</u>, <u>MatrNr</u>, Ausleihdatum)

Umwandlung – Beispiel

4. Der Schlüssel ist nicht vollständig enthalten. Daher ist eine entsprechende Relation hinzuzufügen

   Somit tritt hinzu:

   Entleihe_Exempl(DokNr, MatrNr, Deskriptor)

2.4.2.6 Verlustlosigkeit

- Wir haben in der Vergangenheit verschiedene Schemata in mehrere Schemata zerlegt
- Dabei entstehen aus einer Relation mehrere Relationen
- Die ursprünglichen längeren Tupel müssen somit durch entsprechende Verbundoperationen (Joins) rekonstruiert werden
- Hierbei muss gelten
  - Es dürfen durch die Joins keine Tupel verschwinden
  - Es dürfen keine zusätzlichen Tupel entstehen, die die ursprünglichen Tupel nicht mehr interpretierbar machen
  - Es dürfen keine funktionalen Abhängigkeiten verschwinden

Verlustlosigkeit

- Im Folgenden betrachten wir zunächst das Problem der Erstellung einer Verlustlosen Zerlegung eines Schemas
- Dazu werden wir einen Algorithmus kennen lernen, der eine Zerlegung auf Verlustlosigkeit prüft

Test auf Verlustlosigkeit

Input: $R(A_1,\ldots,A_n)$, F und Zerlegung $\{R_1,\ldots,R_k\}$

1. Bilde Matrix mit k Zeilen und n Spalten. Jede Zeile steht für eine Relation der Zerlegung und jede Spalte für ein Attribut in Relation R
2. Für Zeile i: Trage ein „x“ für jede Spalte j ein, bei der gilt A_j ist Element von R_i, sonst trage ein „o“ ein
3. Nun führe den folgenden Algorithmus aus:
    1. Falls in mindestens zwei Zeilen (d.h. zwei Relationen) die linke Seiten einer Regel von F auftauchen (mit „x“ markiert) dann kann die rechte Seite der Regel ebenfalls in den beiden Zeilen mit „x“ gekennzeichnet werden. Wiederhole das Vorgehen solange bis keine Veränderung mehr möglich ist
    2. Falls im vorherigen Schritt eine komplette „x“-Zeile entsteht gebe „ja“ aus, sonst „nein“

Interpretation

- Der Algorithmus führt einen Join durch und bedient sich der funktionalen Abhängigkeiten um weitere Attribute zu erschließen
- Ist die Zerlegung verlustfrei dann sind per Join alle ursprünglichen Tupel rekonstruierbar
- Das heißt aber, dass eine vollständige „x“-Zeile entsteht

Beispiel – Zwei Zerlegungen

- Spieler(Name, Verein, Sponsor, Vertrag)
- Es gilt
  - Name → Verein
  - Name, Vertrag → Sponsor
- Wir betrachten die Zerlegungen
  - 1: Spieler(Name, Verein, Sponsor) und Geld(Sponsor, Vertrag)
  - 2: Spieler(Name, Verein) und Geld(Name, Vertrag, Sponsor)
- Auf beide Zerlegungen wenden wir unseren Algorithmus an

Beispiel – Erste Zerlegung prüfen

- Im ersten Fall können wir nichts ableiten

Name	Verein	Sponsor	Vertrag
x	x	x	O
o	o	x	x

- Damit ist die Zerlegung nicht verlustfrei

Beispiel – Zweite Zerlegung prüfen

- Im zweiten Fall können wir ableiten

Name	Verein	Sponsor	Vertrag
x	x	o	o
x	o	x	x

- Ableitung der Regel: Name → Verein

Name	Verein	Sponsor	Vertrag
x	x	o	o
x	x	x	x

Name	Verein	Sponsor	Vertrag
x	x	o	o
x	x	x	x

- Damit ist die Zerlegung offensichtlich verlustfrei

Fall 1 – Beispielbelegung

- Ausgangsbelegung

Name	Verein	Sponsor	Vertrag
Müller	FC Bayern München	Coca Cola	176464
Herrmann	Bor. M‘gladbach	Coca Cola	747437
Herrmann	Bor. M‘gladbach	IBM	217476

- Zerlegung

Name	Verein	Sponsor
Müller	FC Bayern München	Coca Cola
Herrmann	Bor. M‘gladbach	IBM
Herrmann	Bor. M‘gladbach	Coca Cola

Sponsor	Vertrag
Coca Cola	176464
Coca Cola	747437
IBM	217476

- Join
  - neue Tupel sind entstanden (kursiv)
  - Die Zerlegung ist **nicht** verlustfrei

Name	Verein	Sponsor	Vertrag
Müller	FC Bayern München	Coca Cola	176464
Müller	*FC Bayern München*	*Coca Cola*	*747437*
Herrmann	*Bor. M‘gladbach*	*Coca Cola*	*176464*
Herrmann	Bor. M‘gladbach	Coca Cola	747437
Herrmann	Bor. M‘gladbach	IBM	217476

Fall 1 – Beispielbelegung

- Ausgangsbelegung

Name	Verein	Sponsor	Vertrag
Müller	FC Bayern München	Coca Cola	176464
Herrmann	Bor. M‘gladbach	Coca Cola	747437
Herrmann	Bor. M‘gladbach	IBM	217476

- Zerlegung

Name	Vertrag	Sponsor
Müller	176464	Coca Cola
Herrmann	747437	IBM
Herrmann	217476	Coca Cola

Name	Verein
Müller	FC Bayern München
Herrmann	Bor. M‘gladbach

- Join
  - Nur die bekannten Tupel entstehen
  - Die Zerlegung ist verlustfrei

Name	Verein	Sponsor	Vertrag
Müller	FC Bayern München	Coca Cola	176464
Herrmann	Bor. M‘gladbach	Coca Cola	747437
Herrmann	Bor. M‘gladbach	IBM	217476

2.4.2.7 Zerlegung – Unlautere Belegungen

- Wir haben gesehen, dass wir Verlustfreiheit bei einer Zerlegung überprüfen können
- Allerdings ist es auch möglich, dass funktionale Abhängigkeiten durch Zerlegungen zerstört werden
- Beispiel hierzu:
  - Biertrinker(Kneipe, Gast, Biersorte)
  - Funktionale Abhängigkeiten
    - Kneipe, Gast → Biersorte
    - Biersorte → Kneipe
  - Wir zerlegen wie folgt (BCNF siehe hierzu Kapitel 2.4.2.9)
    - Biertrinker(Gast, Biersorte)
    - Biersorte(Biersorte, Kneipe)

Verlustfreiheit

- Wir wenden den Test Algorithmus an

Kneipe	Gast	Biersorte
o	x	x
x	o	x

- Wir können die Abhängigkeit (Biersorte → Kneipe) anwenden und erhalten

Kneipe	Gast	Biersorte
x	x	x
x	o	x

- Damit ist die Zerlegung verlustfrei

Beispielbelegung

Ausgangsbelegung

Kneipe	**Gast**	**Biersorte**
Zum Hannes	Bock	Weizenbier
Zum Hannes	Göpfert	Pils
Wildecker	Bock	Kölsch

Und in der Zerlegung

Gast	**Biersorte**
Bock	Weizenbier
Bock	Kölsch
Göpfert	Pils

Kneipe	**Biersorte**
Zum Hannes	Weizenbier
Zum Hannes	Pils
Wildecker	Kölsch

Verbund Bildung

Kneipe	Gast	Biersorte
Zum Hannes	Bock	Weizenbier
Zum Hannes	Göpfert	Pils
Wildecker	Bock	Kölsch

- Wie man sieht, ist alles verlustfrei
- Aber die funktionale Abhängigkeit (Kneipe, Gast → Biersorte) ist zerstört und deshalb in der Zukunft nicht mehr zu überwachen
- Damit kann in Zukunft auch eine neue Belegung in der Relation R(Gast, Biersorte) eingetragen werden, die dann nicht mehr dem Konsum eines Gastes in einer Kneipe entspricht
- Daher ist ein Erhalt der funktionalen Abhängigkeiten in den meisten Fällen sinnvoll

Erhalt der funktionalen Abhängigkeiten

2.4.2.7.1 **Definition**

Sei R ein Relationsschema mit funktionalen Abhängigkeiten F, Attributsmenge $\{A_1,\ldots,A_n\}$ und $Z=\{R_1,\ldots,R_k\}$ eine Zerlegung.

Dann erhält Z die funktionalen Abhängigkeiten F falls gilt:

$$\left(\pi_{R_1}(F)\cup\ldots\cup\pi_{R_k}(F)\right)\Rightarrow F,\ \text{wobei gilt:}$$

$$\pi_{R_i}(F)=\left\{(X\rightarrow Y)\middle|F\Rightarrow(X\rightarrow Y)\wedge X,Y\subseteq R_i\right\}$$

2.4.2.7.2 Algorithmus zur Prüfung

Input: R, F und Zerlegung $Z=\{R_1,\ldots,R_k\}$

Für alle $(X \rightarrow Y)$ aus F tue

Z=X

Solange Z sich verändert tue das Folgende

Von i=1 bis i=k tue

$Z=Z \cup (Z \cap R_i)^+ \cap R_i$ /* Ableitung in F */

Ende tue

Ende Solange

Falls Y Teilmenge von Z dann prüfe weiter sonst verwerfe

Ende tue

Beispiel – Algorithmus zur Prüfung

- Wir betrachten das Beispiel:
  - Spielerverträge(Name, Verein, Vertrag, Sponsor)
  - Funktionale Abhängigkeiten
    - Name → Verein und
    - Verein → Sponsor
  - Zerlegung
    - Spieler(Name, Verein) und Geld(Name, Vertrag, Sponsor)
  - Wir prüfen
    - Name → Verein: Z=Name; Z=Name, Verein ok
    - Verein → Sponsor: Z=Verein; **nicht ok** da Sponsor nicht mehr ableitbar ist
- Damit sind die funktionalen Abhängigkeiten **nicht** erhalten worden

2.4.2.8 Korrektheit des Algorithmus 2.4.2.5.2

- Nun können wir die Korrektheit dieses zentralen Algorithmus‘ zeigen
- Dies geschieht auf den nächsten Folien

Korrektheit des Algorithmus 2.4.2.5.2

1. *Wenn der Algorithmus mit Schritt 2 abbricht, dann ist das Schema bereits in dritter Normalform*

   Beweis:

   Es liegt in diesem Fall eine Regel $X \rightarrow Y$ aus G vor mit $X \cup Y = \{A_1,\ldots,A_n\}$. Da G minimale Überdeckung ist, folgt, dass X ein Schlüssel ist und $|Y|=1$.

   Wir nehmen nun an, dass das Schema nicht in dritter Normalform ist. Dann gibt es ein NSA z mit der Eigenschaft, dass gilt $X \rightarrow Z \rightarrow z$, wobei sich X nicht aus Z ableiten lässt. Ebenso ist z nicht aus Z.

   Da z NSA ist, ist z auch nicht aus X und somit lautet die obige Regel $X \rightarrow Y$ aus G (mit $X \cup Y = \{A_1,\ldots,A_n\}$): $X \rightarrow z$

   Da aber $X \rightarrow Z \rightarrow z$ die Regel $X \rightarrow z$ impliziert kann diese Regel auch in $G/\{X \rightarrow z\}$ hergeleitet werden. Damit ist G aber nicht minimal. Dies ist ein Widerspruch zu Schritt 1. Damit ist das Schema in dritter Normalform.

Korrektheit des Algorithmus 2.4.2.5.2

2. *Schritt 3 liefert Schemata in dritter Normalform*

   Beweis:

   In ein Schema kommen alle Attribute, die aus einer linken Seite ableitbar sind

   Also $(A_1,\ldots,A_k,z_1,\ldots,z_l)$, mit $A_1,\ldots,A_k \rightarrow z_h$ mit $h=1,\ldots,l$

   Wenn nun gelten würde $A_1,\ldots,A_k \rightarrow Z \rightarrow z_h$ ($A_1,\ldots,A_k$ nicht aus Z ableitbar), dann könnte G wiederum verkleinert werden (nämlich um $A_1,\ldots,A_k \rightarrow z_h$). Das wäre aber ein Widerspruch zur Definition von G.

   Also kann es Z nicht geben. Schema ist somit bereits in dritter Normalform.

   Allerdings kann es Attribute geben, die in keiner funktionalen Abhängigkeit auftauchen. Diese werden schließlich durch Schritt 4 erfasst

Korrektheit des Algorithmus 2.4.2.5.2

3. *Schritt 3 erhält die funktionalen Abhängigkeiten*

   Da G äquivalent zu F ist und die Attribute der jeweiligen Regeln in G auch komplett in Relationen Eingang finden, lassen sich alle Regeln in F auch im Relationsschema nachvollziehen. Es werden somit alle Abhängigkeiten aus G (und damit auch aus F) erhalten.

Korrektheit des Algorithmus 2.4.2.5.2

4. *Nach dem Algorithmus ist das Schema verlustlos*

   Wir wenden den Algorithmus aus 2.4.2.6 an. Aufgrund des Schrittes 4 werden alle Attribute Teil der Zerlegung.

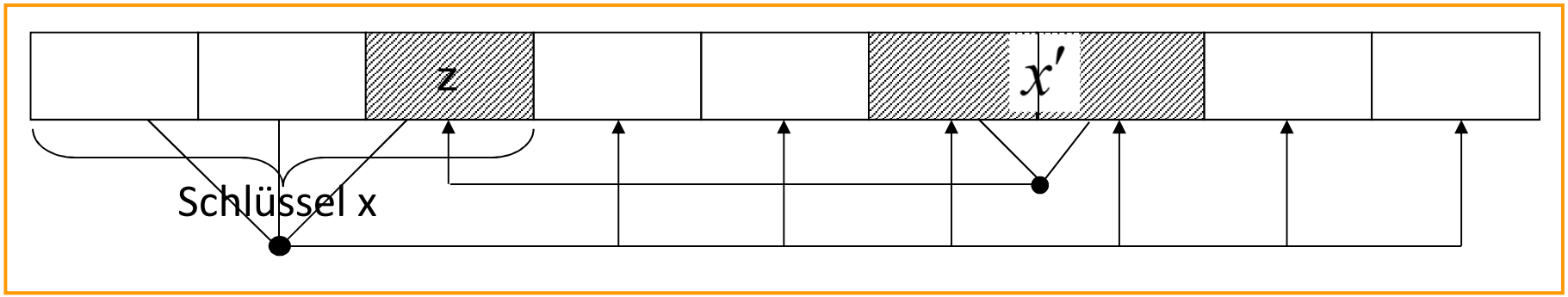
   Nach Abschluss des Verfahrens findet sich immer eine Relation (X,Y) mit X als Schlüssel der ursprünglichen Relation

   Gilt nun $X \cup Y = \{A_1,\ldots,A_n\}$ folgt die Verlustfreiheit durch den Algorithmus direkt nach Ausführung von Schritt 2

   Im anderen Fall wissen wir, dass alle funktionalen Abhängigkeiten der ursprünglichen Relation erhalten worden sind. Daher sind alle Abhängigkeiten $X \rightarrow Y'$ ableitbar. Somit gibt es entsprechende Relationen im Schema und der Algorithmus erzeugt durch wiederholte Anwendung von Schritt 3.1 schließlich eine vollständige „x“-Zeile.

Dritte Normalform – Anomalien

- Leider ist auch die dritte Normalform nicht ohne mögliche Anomalien
- So kann ein Schlüsselattribut von anderen Attributen abhängen
- Trotzdem ist kein Verstoß gegen die Anforderungen der dritten Normalform festzustellen
- Daher gibt es noch weitere (höhere) Normalformen
  - Boyce-Codd-Normalform (ohne Anomalien, aber funktionale Abhängigkeiten können verloren gehen)
  - 4. und 5. Normalform



2.4.2.9 Boyce-Codd-Normalform

2.4.2.9.1 Definition (Determinante)

Gegeben sei in einfaches relationales Schema (R(A_1,…,A_n),F) mit $D \subseteq \{A_1, \ldots, A_n\}$ sowie $A_i \notin D$. Dann heißt D Determinante von A_i genau dann wenn

(1) F impliziert $D \rightarrow A_i$

(2) Falls auch F die Abhängigkeit $C \rightarrow A_i$ mit $C \subseteq D$ impliziert so gilt $C = D$

Das heißt also, dass eine Determinante eine nicht-reduzierbare Menge von Attributen ist, die das Attribut A_i nicht trivial implizieren, d.h. durch funktionale Abhängigkeiten bestimmen.

Wichtige Eigenschaft der 3.Normalform

2.4.2.9.2 Satz (Determinanten und 3.Normalform)

Gegeben sei ein einfaches relationales Schema $(R(A_1,\ldots,A_n),F)$ in dritter Normalform.

Dies ist äquivalent zu der folgenden Aussage:

Alle Determinanten von jedem Nicht-Schlüsselattribut (NSA) des einfachen relationalen Schemas (R(A1,…,An),F) sind Schlüsselkandidaten.

Beweis von Satz 2.4.2.9.2 – Teil 1

- Wir zeigen: Aus der ersten Aussage folgt die zweite:
  - Wir betrachten ein NSA z aus ($R(A_1,\ldots,A_n)$,F) in dritter Normalform und nehmen an, dass eine Determinante D von z kein Schlüsselkandidat ist.
  - Es gibt somit einen Schlüssel Y von ($R(A_1,\ldots,A_n)$,F) mit den folgenden Eigenschaften:
    - $Y \rightarrow D$ und $D \rightarrow z$ mit $z \notin D$.
    - Damit wäre aber ($R(A_1,\ldots,A_n)$,F) nicht in dritter Normalform.
    - Somit muss D ein Schlüsselkandidat sein.

Beweis von Satz 2.4.2.9.2 – Teil 2

- Wir zeigen: Aus der zweiten Aussage folgt die erste:
  - Wir nehmen an, dass alle Determinanten jedes Nicht-Schlüsselattributes (NSA) des einfaches relationales Schemas $(R(A_1,\ldots,A_n),F)$ Schlüsselkandidaten sind aber $(R(A_1,\ldots,A_n),F)$ ist nicht in dritter Normalform
  - Also gibt es ein NSA z mit $X \to Y \to z$ wobei X nicht aus Y ableitbar ist und $z \notin Y$, da die 3. NF verletzt ist
  - Wir bilden minimale Teilmenge $\tilde{Y} \subseteq Y$ mit $\tilde{Y} \to z$ wobei wiederum X nicht aus $\tilde{Y}$ ableitbar ist und $z \notin \tilde{Y}$
  - Dann ist $\tilde{Y}$ Determinante von z und somit (wegen Annahme) ist $\tilde{Y}$ Schlüsselkandidat. Dies aber impliziert $\tilde{Y} \to X$
  - Somit erhalten wir einen Widerspruch zur Annahme dass es die oben definierte Verletzung der 3.NF gibt

Definition der Boyce-Codd Normalform

2.4.2.9.3 Definition (Boyce-Codd Normalform)

Ein einfaches relationales Schema $(R(A_1,\ldots,A_n),F)$ ist in Boyce-Codd Normalform genau dann wenn gilt:

Alle Determinanten von Attributen im Schema (R(A1,…,An),F) sind Schlüsselkandidaten.

Aufgrund von Satz 2.4.2.9.2 gilt:

2.4.2.9.4 Folgerung

Jedes einfache relationale Schema $(R(A_1,\ldots,A_n),F)$ in Boyce-Codd Normalform ist auch in dritter Normalform.

Test auf Einhaltung der Boyce-Codd Normalform

Gegeben: ($R(A_1,\ldots,A_n),F$) und F hat nur einelementige rechte Seiten

Überprüfe für jede funktionale Abhängigkeit $(X \rightarrow A) \in F$ bei der A nicht Teil der Menge X ist ob X ein Superschlüssel ist.

Falls die Aussage für nur eine funktionale Abhängigkeit nicht zutrifft, ist die Boyce-Codd Normalform verletzt.

Algorithmus zur Umwandlung: 3.NF→Boyce Codd NF

2.4.2.9.4 Algorithmus

Input: Schema S in der dritten Normalform

Output: Schema in Boyce-Codd Normalform

1. Solange S noch nicht in Boyce-Codd NF
   1. Nehme Relation R, die die Boyce Codd NF wegen $X \rightarrow Y$ verletzt heraus (dabei gilt X∩Y=∅)
   2. Ersetze diese Relation durch die Relationen $R - Y$ und $X \cup Y$ mit den funktionalen Abhängigkeiten von S, die nur die jeweiligen Attribute umfassen
2. Gebe S aus

Korrektheit des Algorithmus 2.4.2.9.4

- Da in jedem Schritt eine Verletzung der Boyce-Codd Normalform entfernt wird ohne eine neue zu produzieren wird diese Normalform nach einer endlichen Zahl von Durchläufen erreicht
- Im Folgenden zeigen wir noch, dass die durchgeführten Aufteilungen der Relationen verlustfrei sind
- Wir haben allerdings schon gesehen, dass hierdurch funktionale Abhängigkeiten verloren gehen können

Korrektheit des Algorithmus 2.4.2.9.4

2.4.2.9.5 Lemma

Die Zerlegungen im Algorithmus 2.4.2.9.4 sind verlustlos

Beweis:

- Wir wenden den Algorithmus aus Kapitel 2.4.2.6 an
- Relationen R-Y und XUY mit X →Y

Relation	(R-Y)-X	X-Y=X	Y
R-Y	x	x	O
XUY	o	x	x

- Durch Anwendung der funktionalen Abhängigkeit X →Y entsteht

Korrektheit des Algorithmus 2.4.2.9.4

Relation	(R-Y)-X	X-Y	Y
R-Y	x	x	x
XUY	o	x	x

- Damit entsteht in der ersten Relation eine vollständige Zeile
- Die Zerlegung ist somit verlustlos

Beispiel – Zwei Zerlegungen

- NON-BCNF(A,B,C,D)
- Es gilt
  - A,B → C,D
  - C,D → A (verletzt die BCNF)
- Zerlegung nach Algorithmus 2.4.2.9.4
  - BCNF1(B,C,D) und BCNF2(A,C,D)
  - Aber A,B → C und A,B → D sind nicht mehr verfolgbar
- Es geht aber auch alternativ
  - BCNF1(A,C,D)
  - BCNF2(A,B,D)
  - BCNF3(A,B,C)

Beispiel – Zerlegungen

Gegeben: R = {A,B,C} und F = {{A,B} → {C}, {C} → {B}}.

- R ist in 3NF aber nicht in BCNF, da C Determinante von B, aber C kein Schlüsselkandidat
- Die einzigen drei mögliche Zerlegungen sind:

a) $R_1 = \{A,B\}$, $R_2 = \{B,C\}$

A	B	C
x	x	
	x	x

nicht verlustfrei

b) $R_1 = \{A,B\}$, $R_2 = \{A,C\}$

A	B	C
x	x	
x		x

nicht verlustfrei

c) $R_1 = \{A,C\}$, $R_2 = \{B,C\}$

A	B	C
x	x	x
	x	x

verlustfrei

- Aber: {A,B} → {C} nicht folgerbar