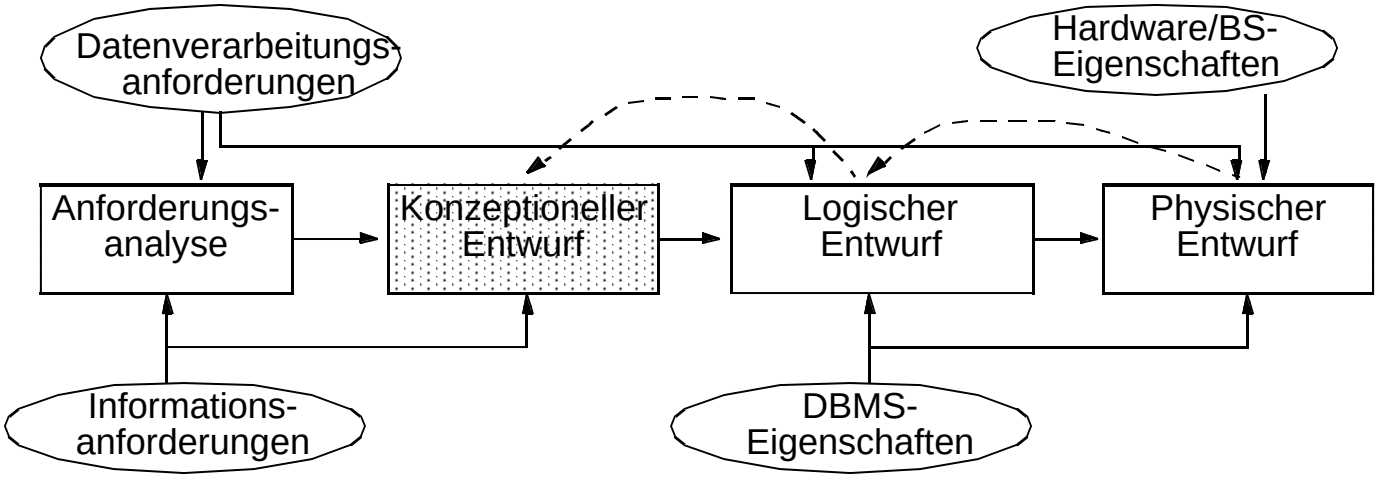
4. Datenbankentwurf

- **Bisher in der Vorlesung**
  - Fokussierung auf Anfragen an Datenbanken
    - Relationale Algebra / Tupelkalkül
    - SQL

- **Vorher muss Datenbank angelegt werden!**
  - Wie entwerfe ich ein Datenbankschema?
    - Gibt es ein Vorgehensmodell?
  - Gibt es Qualitätskriterien für den Entwurf?
    - Wie viele Relationen(schemata) pro Datenbank?
  - Welche Indexe sollten angelegt werden?

Datenorientierter Ansatz

- **Erstellung eines Entwurfs in mehreren Schritten**



Anforderungsanalyse

- Problemstellung aus dem Bereich der Softwaretechnik
  → siehe Vorlesungen Bockisch/Taentzer
- Die Analyse basiert auf dem Wissen über
  Informations(struktur)anforderungen, z. B.
  - Was sind meine Objekte und deren Attribute?
  - Wie sehen die Beziehungen zwischen Objekten aus?
  - Wie viele Objekte werden in meiner Datenbank auftreten?

  und **Datenverarbeitungsanforderungen, z. B.**
  - Was sind typische Prozesse?
  - Reihenfolge und Priorität der Operationen

Pflichtenheft

- **Zentrales Problem bei der Anforderungsanalyse:**
  - Entwickler einer Datenbank muss diese Informationen erst von den Benutzern der Datenbank bekommen!
    → Benutzerbefragungen
  - Es gibt kein Patentrezept für eine erfolgreiche Anforderungsanalyse.

- **Pflichtenheft** ist das Resultat der Anforderungsanalyse
  - **Was hat die Datenbank zu leisten?**
    - Informationsanforderungen
    - Datenverarbeitungsanforderungen
  - **Grundlage bei der Vergabe eines Auftrags und bei späteren Streitigkeiten**

Konzeptioneller Entwurf

- **Ziel**
  - Erstellung eines konzeptionellen Modells unabhängig
    - vom konkreten DBMS
    - und von einem konkreten Datenbankmodell.
- **Methode**
  - Datenbeschreibung in einer **formalen Sprache** auf Basis eines Modells hoher Abstraktion
    - UML (Unified Model Language) → siehe Softwaretechnik
    - Entity-Relationship Modell (ER-Modell)
- **Problem**
  - KEINE automatische Transformation der Anforderungsanalyse in ein konzeptionellen Entwurf möglich
    - Informelle Anforderungsanalyse ←→ formales konzeptionelles Modell
    - Weglassen irrelevanter Strukturen (Abstraktion realer Objekte)

Logischer Entwurf

- **Voraussetzung**
  - Festlegung des logischen Datenmodells
- **Ziel**
  - Abbildung der Datenstrukturen des konzeptionellen Modells in Datenstrukturen des darunter liegenden logischen Datenmodells.
- **Beispiel**
  - Transformation: ER-Modell → relationales Modell
    - Möglichst kompakte Repräsentation Daten (Vermeidung von Redundanz)
- **Abstraktionsgrad**
  - Datenstrukturen des logischen Modells sind unabhängig von ihrer physischen Repräsentation.
  - Konkretes DBMS hat keinen Einfluss auf Modellierung

Physischer Entwurf

- **Voraussetzung**
  - Logischer Entwurf in Form eines relationalen Modells
- **Ziel**
  - Physische Repräsentation der Datenstrukturen des logischen Entwurfs.
- **Beispiele**
  - Aufteilung der Datenbank auf verschiedene Speichersysteme (zur Lastbalancierung)
  - Anlegen von Hilfsstrukturen wie z. B. Indexe zur Unterstützung von Anfragen.
    - Zu viele Indexe → Updates werden zu teuer
    - Zu wenige Indexe → schlechte Anfrageleistung

4.1. Entity-Relatioship Datenmodell

- **Kurz ER-Modell**
  - Peter P. Chen: The Entity-Relationship Model - Toward a Unified View of Data. in Trans. on Database Systems 1(1): 9-36(1976))

- **Ziel**

  Modellierung eines Ausschnittes der “realen Welt” durch **Abstraktion**, so dass gewisse Fragen über die “reale Welt” mit Hilfe des Modells beantwortet werden können.
  - *“Reale Welt” ist zunächst nur wahrnehmbar über Sinnesorgane. Menschliche Sprache ist bereits erster Abstraktions- und Modellierungsschritt.*

- **Vorgehensweise beim DB-Entwurf (siehe oben)**
  - Zuerst: Anforderungsanalyse und Entwurf des ER-Modells
  - Dann: Umsetzung des ER-Modells in das logische Datenbankmodell

Grundlagen

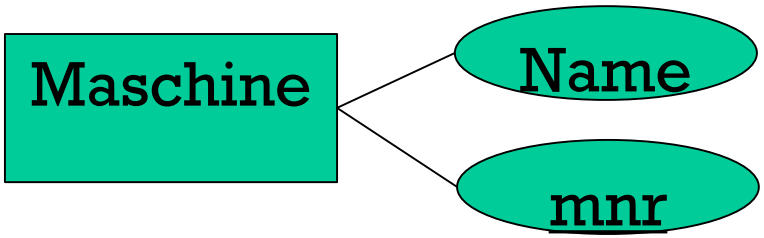
- **ER-Modell beschreibt “reale Welt” durch**
  - **Entitäten** (*Entities) mit*
  - **Eigenschaften** (*Attributes) und*
  - **Beziehungen** (*Relationships) zueinander.*

Entität und Entitätstyp

- Eine **Entität** existiert in der realen, zu modellierenden Welt und unterscheidet sich von anderen Entitäten.
  - Beispiel: Angestellter, Maschine, …
- Strukturgleiche Entitäten werden zu einem **Entitätstypen** (auch Entitätsmengen) zusammengefasst.
  - Menge aller Angestellten, Menge aller Maschinen, Menge aller Abteilungen
  - Eine Entitätstyp enthält alle möglichen Entitäten.
    - unabhängig vom den derzeitigen Instanzen.
- Ein Entitätstyp wird durch die zugehörigen **Attribute** und weitere semantische Eigenschaften (→ **Integritätsbedingungen**) beschrieben.
  - Ein Attribut ist eine charakteristische Eigenschaft.
  - Jeder Angestellte besitzt eine Personalnummer
- Eine minimale Menge von Attributen, anhand deren Werte sich alle Entitäten eines Entitätstyps unterscheiden lassen, wird als **Schlüssel** bezeichnet.
  - z.B. ISBN-Nummer ist Schlüssel für den Entitätstyp Buch

Graphische Repräsentation

- **Entitätstypen**
  - Rechtecke
  - Name des Entitätstyp
- **Attribute**
  - Ellipse, die mit dem Rechteck des Entitätstyps verbunden ist.
  - Name des Attributs
  - Schlüssel wird unterstrichen



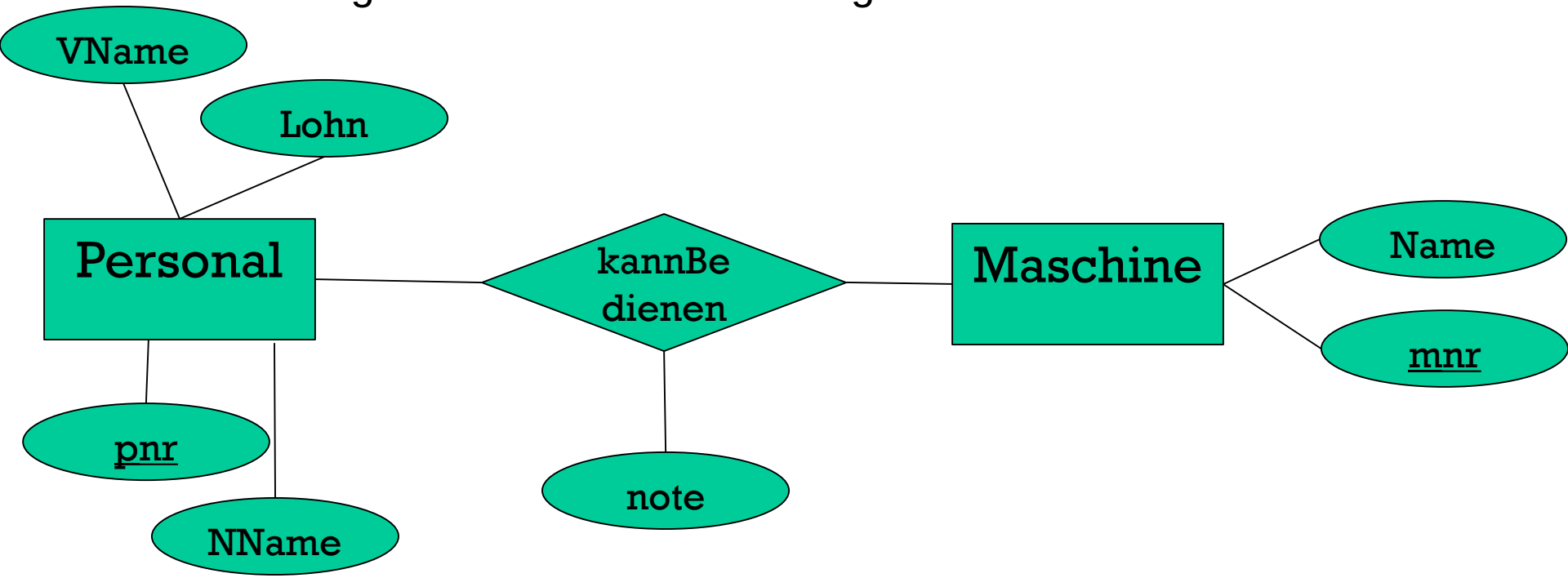
Beziehung und Beziehungstyp

- Eine **Beziehung** repräsentiert Zusammenhänge zwischen Entitäten.
  - *Beispiele:*
    - *Angestellter mit Personalnummer 9876 kann die Maschine mit Nummer 1234 bedienen.*

- Eine Menge von strukturgleichen Beziehungen wird zu einem **Beziehungstyp** $R(E_1,\ldots,E_n)$ *(Beziehungsmenge) zusammengefasst.*
    - z. B. die Beziehung *kannBedienen*
  - Ein Beziehungstyp besteht aus
    - n Entitätstypen, n > 1, (n = **Grad** des Beziehungstyps)
    - und zusätzlichen Attributen.
  - Ein Entitätstyp darf in einem Beziehungstyp mehrfach vorkommen.
    - Zur Unterscheidung werden dann **Rollen** an die Entitätstypen vergeben.

Graphische Repräsentation

- **Beziehungstypen**
  - Rauten
  - Name des Beziehungstyps
  - Verbindung mit den entsprechenden Entitätstypen
    - Ggf. steht an der Verbindungslinie der Rollenname



Funktionalität von Beziehungstypen

- **Ziel**
  - Bessere semantische Charakterisierung von Beziehungstypen

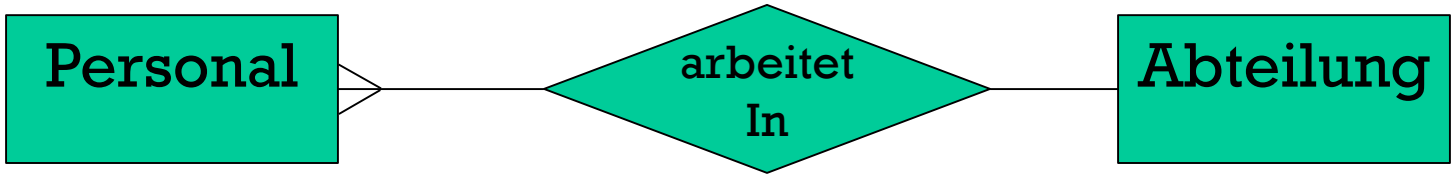
- **Klassische Unterscheidung**
  - Annahme: zweistelliger Beziehungstyp **$R(E_1,E_2)$**
  - **1:1-Beziehungstypen** (one-to-one relationships)
    - Falls jede Entität aus E_1 zu höchstens einer Entität aus E_2 in Beziehung steht und umgekehrt.
  - **1:M-Beziehungen** (one-to-many relationships)
    - Falls jede Entität aus E_1 mit beliebig vielen (also mehreren oder auch keinen) Entitäten aus E_2, aber jede Entität aus E_2 mit maximal einer Entität aus E_1 in Beziehung steht.
  - **M:N-Beziehungen** (many-to-many relationships)
    - Falls jede Entität aus E_1 mit beliebig vielen (also mehreren oder auch keinen) Entitäten aus E_2 in Beziehung stehen kann und umgekehrt.

Graphische Repräsentationen

- **Orginalvorschlag**

- **Funktionale Repräsentation**

- **Krähenfußnotation**



Min-max-Notation

- **min-max-Notation**
  - Zusätzlich zu der Unterscheidung der Funktionalität hat sich die min-max-Notation bewährt.
- **Definition**
  - Seien E_1 und E_2 Entitätstypen und $R(E_1,E_2)$ ein zweistelliger Beziehungstyp.
    - Eine Kante von R zum Entitätstyp E_i wird mit (min_i,max_i) i=1,2 annotiert. Dabei gilt:
      - für alle $e_1 \in E_1$ gilt $min_1 \leq |\{(e_1,e_2) \mid e_2 \in E_2\}| \leq max_1$
      - für alle $e_2 \in E_2$ gilt $min_2 \leq |\{(e_1,e_2) \mid e_1 \in E_1\}| \leq max_2$
    - Wenn es keine obere Schranke gibt (oder diese unbekannt ist), wird dies durch ein “*” gekennzeichnet.

Graphische Repräsentation

- **Beispiel**

  - Welche Aussagen kann man aus der min-max-Notation ableiten?

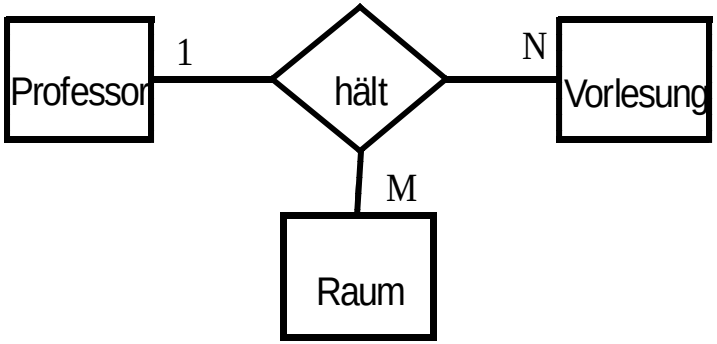
Beziehungen mit mehr als zwei Entitätstypen (1)

- **Funktionale Notation (Verallgemeinerung)**
  - Im Folgenden sei $R(E_1, \ldots, E_n)$ ein Beziehungstyp mit n Entitätstypen $E_1, \ldots, E_n$, $n > 1$. Wir ordnen in unserer Notation der Kante von R zu E_j eine "1" zu, falls

$$R: (E_1, \ldots, E_{j-1}, E_{j+1}, \ldots, E_n) \rightarrow E_j$$

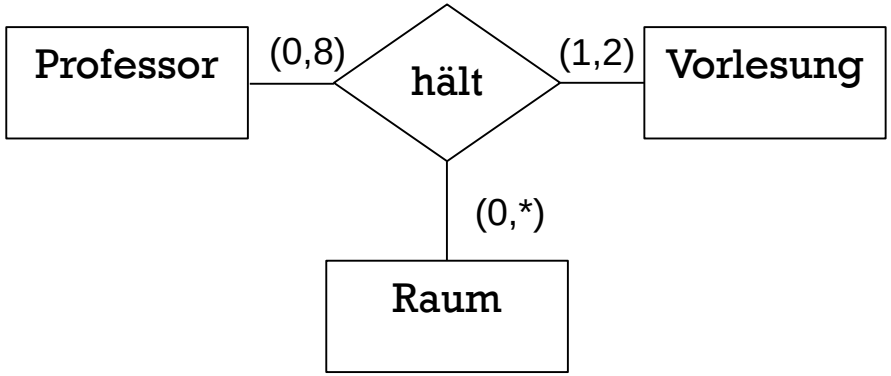
    eine **Funktion** ist. Ansonsten wird einer Kante ein Symbol M,N, … zugeordnet.

- **Beispiel**



Beziehungen mit mehr als zwei Entitätstypen (2)

- **min-max-Notation (Verallgemeinerung)**
  - Sei ein Beziehungstyp $R(E_1,\ldots,E_n)$ mit n Entitätstypen $E_1,\ldots, E_n$, $n > 1$ gegeben.
    - An einer Kante von R zum E_j wird (min_i,max_i) $j=1,\ldots,n$ notiert. Dabei gilt:
      - Für alle $\mathbf{e}_j \in E_j$ gilt: $min_j \leq |\{(e_1,...,e_{j-1},e_{j+1}, ..., e_n) | (e_1,...,e_{j-1},\mathbf{e_j}, e_{j+1}, ..., e_n) \in R\}| \leq max_j$
    - Wenn es keine obere Schranke gibt (oder diese unbekannt ist), wird diese durch ein "*" gekennzeichnet.
  - Beispiel



Id-Beziehungen

- **Id-Beziehungen** sind spezielle 1:N-Beziehungen, bei denen die Existenz einer Entität von einer anderen Entität abhängt.
  - Man bezeichnet dann auch den existenzabhängigen Entitätstyp als **schwach** und den anderen Entitätstyp als **stark**.
- Graphische Notation
  - Doppelraute und Doppelrechteck (Schwacher Entitätstyp)

IS-A-Beziehungstypen

- **IS-A-Beziehungstyp** (auch Typerweiterung)
  - Eine Entität vererbt alle ihre Eigenschaften an eine andere Entität. Die Beziehung zwischen den Entitätstypen wird als **IS-A-Beziehung** bezeichnet.
    - IS-A-Beziehung wird für die Partitionierung einer Menge in (disjunkte) Teilmengen verwendet.
    - Beide Entitätstypen einer IS-A-Beziehung besitzen den gleichen Schlüssel.
- **Beispiele**
  - ANGSTELLTE besitzen Attribute pnr, nname und lohn
  - VERWALTUNGSANGESTELLTE
    - zusätzliches Attribut: Ressort.
  - PRODUKTIONSANGESTELLTE
    - zusätzliches Attribut: Ausbildung

4.2 Abbildung ER-Modell in ein relationales Datenmodell

- **Datenstrukturen der ER-Datenmodellierung**
  - Entitätstypen
  - Beziehungstypen
- **Datenstruktur des relationalen Modells**
  - Relationen (bzw. Relationenschema)

Fragestellung

- Wie kann ein ER Datenmodell in ein relationales Model umgesetzt werden?
- Diese Frage wird nun in zwei Schritten beantwortet:
  - Einfache Umsetzung von Entitätstypen und Beziehungstypen
  - Konsolidierung des Relationenschemas

Entitätstypen

- **Jeder Entitätstyp wird fast 1:1 als eigenständige Relation umgesetzt, wobei**
  - Jedes Attribut aus dem Entitätstyp auf ein Attribut des Relationenschemas abgebildet wird.
    - Namen können dabei übernommen werden.
  - Der Schlüssel des Entitätstyps wird zum Primärschlüssel des Relationenschemas.

- **Beispiel**
  - Entitätstyp Maschine(mnr: Integer, name: String)

    → Relation Maschine(mnr: Integer, name: String)

Beziehungstyp

- Jeder Beziehungstyp wird zu einer eigenständigen Relation. Dabei gilt:
  - Jedes Attribut des Beziehungstyps wird als ein Attribut in der Relation dargestellt.
  - Die Primärschlüssel der beteiligten Entitäten werden (→ Fremdschlüssel) in das Schema übernommen.
    - Für N:M-Beziehungstypen bilden diese dann auch den Primärschlüssel der Relation.
    - Attribute der Fremdschlüssel müssen ggf. noch umbenannt werden → Eindeutigkeit der Attributnamen
- **Beispiel (M:N-Beziehungstyp)**
  - kannBedienen(Personal, Maschine) mit Attribut note → Relation PMRZuteilung(pnr, mnr, note)

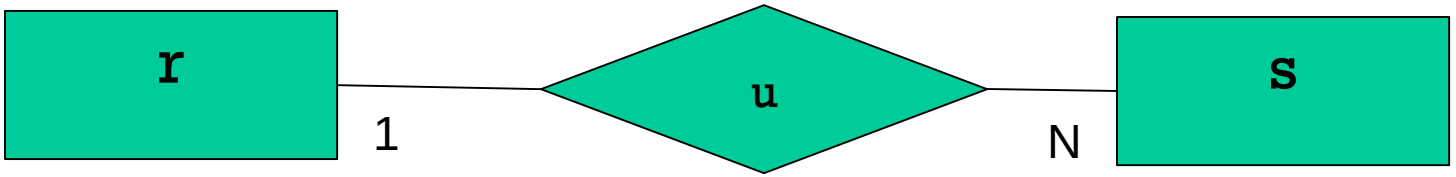
Beispiel (1:N-Beziehung)

- **Beispiel**
  - Beziehungstyp arbeitetIn(Angestellte, Abteilung)
  - Nach unsrem bisherigen Vorgehen würde daraus eine neue Relation arbeitetIn(pnr, abtnr) enstehen.
  - Was wäre der Primärschlüssel dieser Relation?

Konsolidierung (1)

- **Ziel**
  - Vereinfachung des Datenbankschemas durch Verschmelzen von Relationen
    - Einschränkung auf 1:1, 1:N und N:1 Beziehungstypen

- **Notation**
  - Seien r und s die Relationen der beteiligten Entitätstypen eines 1:N-Beziehungstyps und u die Relation des Beziehungstyps.



r(pkr, …) u(pks,pkr, …) s(pks, …)

Konsolidierung (2)

- **Konsolidiertes Datenbankschema**
  - Die Relationen s und u mit gleichem Primärschlüssel werden durch die Relation v mit
    - v = **left-outer-join**(s,u) ersetzt.
  - Die Relation r bleibt unverändert bestehen.

- **Was sind die Vor- und Nachteile einer solchen Konsolidierung?**

4.3 Entwurfstheorie relationaler Datenbanken

- **Fragen**
  - Haben wir durch das Verfahren aus Abschnitt 4.2 ein gutes Datenbankschema gewonnen?
  - Wie kann die Güte eines Datenbankschemas beurteilt werden?
- **Mögliche Indikatoren für die Güte**
  - **Korrektheit**
    - Sind alle Sachverhalte der realen Welt in einem Schema auch entsprechend abgebildet?
  - **Speicherplatzbedarf**
    - Wichtig in der Praxis!
  - **Zugriffszeit** beim Beantworten von **Anfragen**
    - Benutzer sollten möglichst sofort die Ergebnisse bekommen.
  - **Zeit** und Ressourcenbedarf für das **Ändern** der Daten
    - Möglichst viele Änderungen innerhalb einer Sekunde
- **Gewichtung der Kriterien nach Anwendungstyp**

Beispiel

- **In einer Datenbank sollen Kunden, Aufträge und Lieferanten modelliert werden.**
  - Dabei wurden zwei unterschiedliche Entwürfe (**rot** und **blau**) für ein Datenbankschema vorgeschlagen.

Schema rot:

Kunde *(KName, KAdr, Kto)*

Auftrag (KName, Ware, Menge)

Lieferant W(LName, LAdr, Ware, Preis)

?

Schema blau:

KundenAdr (KName, KAdr)

KundenKto (KName, Kto)

Auftrag (KName, Ware, Menge)

Lieferant (LName, LAdr)

Angebot (LName, Ware, Preis)

Redundanz in den Daten

- **Betrachten wir Relation LieferantW (LName, LAdr, Ware, Preis) aus dem Schema rot.**
  - SQL-Befehl

    create table LieferantW(LName varchar, LAdr varchar not null,
    Ware varchar, Preis float,
    primary key (LName, Ware));
  - Inhalt der Tabelle

LieferantW	LName	LAdr	Ware	Preis
	Müller	München	Milch	1,10
	Kohl	Frankfurt	Milch	1,30
	Kohl	Frankfurt	Mehl	2,10
	Keller	Stuttgart	Mehl	2,20

  - Für jede Ware, die ein Lieferant liefert, wird die Adresse des Lieferanten gespeichert.
    - Mehrmalige Speicherung der Adresse ➔ Redundanz

Anomalien

LieferantW	LName	LAdr	Ware	Preis
	Müller	München	Milch	1,10
	Kohl	Frankfurt	Milch	1,30
	Kohl	Frankfurt	Mehl	2,10
	Keller	Stuttgart	Mehl	2,20

- **Auf Grund dieser Redundanz ergeben sich folgende Anomalien.**
  - **Änderungsanomalie**
    - Die Adresse eines Lieferanten kann in einem seiner Tupel geändert werden und in den anderen nicht.
  - **Einfügeanomalie**
    - Eine Lieferantenadresse kann nur mit einer Ware eingefügt werden.
  - **Entfernungsanomalie**
    - Beim Löschen der letzten Ware geht auch die Lieferantenadresse verloren.

Verhinderung von Anomalien (1)

- **Lassen sich diese Probleme dadurch beheben, dass eine Relation in zwei oder mehre Relationen aufgeteilt wird?**
  - Beispiel:
    - Verbesserung des Entwurfs durch blaues Schema?

      Lieferant (LName, LAdr)

      Angebot (LName, Ware, Preis)

Angebot	LName	Ware	Preis
	Müller	Milch	1,10
	Kohl	Milch	1,30
	Kohl	Mehl	2,10
	Keller	Mehl	2,20

Lieferant	LName	LAdr
	Müller	München
	Kohl	Frankfurt
	Keller	Stuttgart

- **Vorteil**
  - **Keine Redundanz und Anomalien ➔ Niedrige Kosten bei Änderungen**
- **Nachteil**
  - **Zusätzliche Joinoperationen bei Anfragen ➔ Hohe Anfragekosten**

Verhinderung von Anomalien (2)

- **Fragen:**
  - Hilft eine solche Zerlegung immer ?
  - Wie erkennt man zu trennende Attribute?
  - Müssen alle Attribute getrennt gespeichert werden ?

- **Beispiel**
  - Ist die folgende Zerlegung auch sinnvoll?
    - Lieferant2 (LName, LAdr,Ware)
    - Angebot2 (Ware, Preis)
  - Erzeugen der Relationen
    - Lieferant2 = $\pi_{\{LName,LAdr,\ Ware\}}$ (LieferantW)
    - Angebot2 = $\pi_{\{Ware,\ Preis\}}$ (LieferantW)
  - Jedoch können wir aus diesen beiden Relationen nicht mehr die ursprüngliche Relation LieferantW erzeugen!

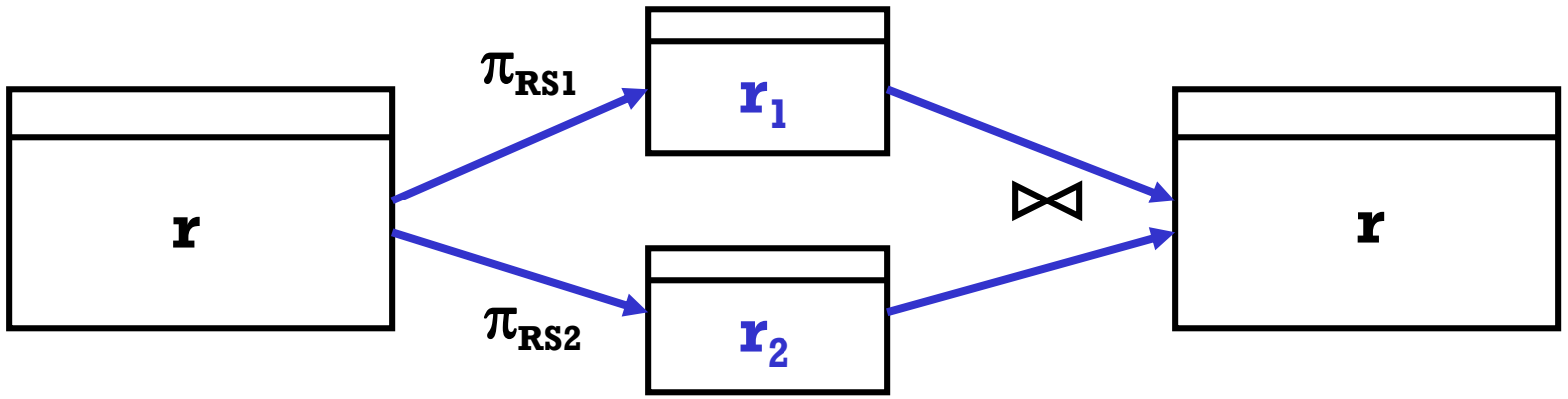
Informationsverlust

- Forderung an eine Zerlegung
  - Eine Zerlegung des Relationsschemas muss so erfolgen, dass alle Zustände der zerlegten Relationen über genau dieselben Informationen wie die ursprüngliche Relation verfügen.
- Definition (**Zerlegung**)
  - Unter einer Zerlegung einer Relation r mit Schema RS_r verstehen wir eine Aufteilung in zwei Relationen r_1 und r_2 mit Schemata RS_1 und RS_2, so dass

$$RS_r \subseteq RS_1 \cup RS_2$$

    gilt.
- Definition (**Verlustlose Zerlegung**)
  - Eine Zerlegung einer Relation r in zwei Relationen r1 und r2 ist verlustlos, falls

$$r = r_1 \bowtie r_2$$



Entwurfsziele

- Statt den Datenbankentwurf mit einem ER-Diagramm informell durchzuführen, soll ein Algorithmus ein gutes Schema automatisch erzeugen.
  - Optimierung im Algorithmus auf Basis formaler Kriterien
    1. Vermeidung von Redundanzen und Anomalien
    2. Vermeidung des Informationsverlustes
    3. evtl. Einbeziehung von Effizienzüberlegungen

- **Grundlagen hierfür**
  - funktionale Abhängigkeiten (Definition folgt)
  - Normalformen

Vorgehensweisen

- **Zwei Varianten bei der algorithmischen Erstellung eines guten Datenbankschemas**
  - **Top-down**
    - Am Anfang besteht das Schema aus nur einer Relation
      - Alle erfassten Attribute liegen im Relationenschema.
    - Iteratives Zerlegen des gegebenen Datenbank-Schemas in ein äquivalentes Schema ohne Redundanz und Anomalien (“Normalisierung”).
  - **Bottom-up** (Synthese)
    - Iteratives Erzeugen der Zielrelationen

4.3.1 Funktionale Abhängigkeiten

- **Funktionale Abhängigkeiten** (**functional dependencies** oder **FD**) sind statische Integritätsbedingungen
  - ➔ **Semantische Bedingungen**, um die Menge der Datenbankzustände einzuschränken.
- FDs sind Teil der Informationsanforderungen und werden in Absprache mit dem Anwender bei der **Anforderungsanalyse** gewonnen.

- **Definition (funktionale Abhängigkeit):**
  - Seien *A, B* $\subseteq$ *RS Teilmengen eines Relationenschemas RS. B ist von A* ***funktional abhängig*** *oder A bestimmt B funktional, geschrieben A* ➔ *B, genau dann wenn zu jeder Relation r* $\in$ *REL(RS) und jedem Wert in A genau ein Wert in B gehört:*
    - Für alle $t_1, t_2 \in r$: $t_1[A] = t_2[A] \Rightarrow t_1[B] = t_2[B]$

Beispiel

- **Relationenschema**
  - Lieferant(LName, LAdr, Ware, Preis)
  - Funktionale Abhängigkeiten

    (FD_1) {LName} → {LAdr}
    - Interpretation:
      - ein Lieferantenname bestimmt eindeutig seine Adresse

    (FD_2) {LName, Ware} → {Preis}
    - Interpretation
      - {LName, Ware} bestimmt eindeutig den Preis

    (FD_3) {LName} → {LName} (trivial)

    (FD_4) {LName, Ware} → {Ware} (trivial)

    (FD_5) {LName, Ware} → {LAdr} (partiell)

Überprüfen von FDs (1)

- **Funktionale Abhängigkeiten sind wichtige Regeln, um die Datenqualität sicherzustellen.**
  - Soll in einer Relation r die FD A→B überprüft werden, muss folgende SQL-Anfrage **kein** Ergebnis liefern:

```
select A, count( distinct B)
from r
group by A
having (count(distinct B) > 1)
```

- **Diese Überprüfung garantiert aber nicht, dass auch zukünftig die FD erfüllt ist.**

Überprüfen von FDs (2)

- **Funktionale Abhängigkeiten sind wichtige Regeln, um die Datenqualität sicherzustellen.**
  - Beim Einfügen eines neuen Datensatzes (…., a, …., b,…) in die Relation r muss vorher überprüft werden, ob die folgende Anfrage leer ist:

```
select *
from r
where A = a and B <> b
```

  - Dies kann mit logarithmischem Aufwand (in der Anzahl der Tupel in der Relation r) erfolgen, wenn ein Index auf dem Attribut A existiert.

Ziel

- **Hohe Datenqualität**
  - Erfassung aller funktionaler Abhängigkeiten in einer Anwendung.
    - → Menge F von FDs
- **Geringe Kosten, um die Datenqualität zu prüfen.**
  - Anlegen eines Index für jede FD
  - Zwar kann jede FD in logarithmischer Zeit geprüft werden, aber die Gesamtkosten sind sehr hoch.

➔ **Minimierung der Anzahl der FDs**

- Gibt es eine zu F äquivalente Menge F_c, die
  - mit weniger FDs als in F auskommt
  - und jede FD aus F auch aus F_c abgeleitet werden kann?

Vorgehensweise

- **Ableitung aller FDs für eine vorgegebene Menge F**
  - ➔ Hülle F^+
- **Effiziente Überprüfung, ob eine FD A➔ B aus einer vorgegebenen Menge F ableitbar ist.**
- **Effiziente Erstellung einer minimalen Repräsentantenmenge F_c, so dass alle FDs in F aus F_c ableitbar sind.**
  - ➔ Beim Einfügen eines neuen Datensatzes genügt es die Regeln in F_c zu überprüfen.

Besondere FDs

- **Definitionen**
  - Eine FD A→B ist **trivial**, wenn gilt B $\subseteq$ A.
  - Eine FD A→B heißt **voll**, *wenn es keine echte Teilmenge C $\subseteq$ A gibt, so dass C → B gilt.*

    Gibt es eine solche Teilmenge C, dann heißt A → B ***partielle*** *Abhängigkeit.*
  - Seien A, B $\subseteq$ RS und A → B (aber nicht B → A). Sei X $\in$ RS - (A $\cup$B) und gelte B → {X}. Dann ist *{X}* **transitiv** *abhängig von A: A → {X}.*

Berechnung von FDs

- **Ziel**
  - Zu einer gegebenen Menge F von FDs soll eine möglichst kleine **äquivalente** Menge F_c von FDs berechnet werden.

- **Vorgehensweise**
  1. Berechne aus F die Menge F^+ aller daraus ableitbaren FDs.
     - Wie kann man aus F neue FDs ableiten?
  2. Verkleinere F so, dass aus der reduzierten Menge immer noch F^+ abgeleitet werden kann.

Hülle einer Menge von FDs

- Zu einer gegebenen Menge F von FDs soll F^+, die Menge aller gültigen FDs berechnet werden.
  - F^+ wird die **Hülle** **von F** bezeichnet.
- Zur Berechnung von F^+ werden folgende Regeln genutzt (**Amstrong Axiome**):
  - ***Reflexivität***: *Sei* $B \subseteq A$. *Dann gilt stets* $A \rightarrow B$.
  - ***Verstärkung***: *Falls* $A \rightarrow B$ *gilt, dann gilt auch* $A \cup C \rightarrow B \cup C$.
  - ***Transitivität***: *Falls* $A \rightarrow B$ *und* $B \rightarrow C$, *dann gilt auch* $A \rightarrow C$
- **Theorem**
  - Die Amstrong Axiome sind korrekt und vollständig!
    1. Alle abgeleitete Regeln sind gültig.
    2. Alle gültigen FDs in F^+ können mit Hilfe dieser Regeln auch hergeleitet werden.

Ableitungsregeln

- Trotz dieser Eigenschaften der Amstrong-Axiome ist es komfortabler noch folgende Regeln zu benutzen:
  - ***Vereinigungsregel***

    Falls $A \rightarrow B$ und $A \rightarrow C$ gilt, dann gilt auch $A \rightarrow B \cup C$
  - ***Dekompositionsregel:***

    Falls $A \rightarrow B \cup C$ gilt, dann gilt auch $A \rightarrow B$ und $A \rightarrow C$
  - ***Pseudotransivität:***

    Falls $A \rightarrow B$ und $C \cup B \rightarrow D$ gilt, dann gilt auch $A \cup C \rightarrow D$

- Beispiel
  - Relation ***Lieferant(LName, LAdr, Ware, Preis)***
  - FD_1-FD_4 seien gültig (siehe oben)
  - zu zeigen: FD_5 {LName, Ware} → {LAdr} ist erfüllt.

Membership-Problem

- **Fragestellung**
  - Sei F eine Menge von FDs und A→B eine FD nicht notwendigerweise aus F. Gilt A → B ∈ F+ ?
    - **Explizite Berechnung von F^+ ist zu aufwendig!**
    - Stattdessen wird die **Hülle A^+ der Attributmenge A** bzgl. der Menge F berechnet.
      - A^+ besteht aus allen Attributen, die von A funktional bestimmt werden.
      - Falls $B \subseteq A^+$ gilt, dann gilt auch $A \rightarrow B \in F^+$.

- **Algorithmus Hülle(F, A)**

```
Erg = A;                                        // Es gilt A → A.
WHILE (Änderungen bei Erg)
        FOREACH  B→C ∈ F  DO
                IF (B ⊆ Erg) Erg = Erg ∪ C ;
RETURN Erg;
```

Kanonische Überdeckung (1)

- **Definition**

  Zwei Mengen F und G von FDs zu einer Relation r sind äquivalent, falls $F^+ = G^+$ gilt.

- **Ziel**
  - Zu gegebenem F berechne eine möglichst kleine äquivalente Menge.

    ➔Minimierung des Aufwands zum Testen der FDs beim Einfügen neuer Tupel in die Relation.

Kanonische Überdeckung (2)

- **Definition**
  - F_c wird als **kanonische Überdeckung** von F bezeichnet, falls folgende Bedingungen erfüllt sind:
    - $F_c^+ = F^+$
    - Für alle FDs $A \rightarrow B$ aus F_c gibt es keine "überflüssigen" Attribute auf der linken und der rechten Seite, d. h.
      - für alle Attribute $X \in A$ gilt

        $$(F_c - \{A \rightarrow B\} \cup \{(A-X) \rightarrow B\})^+ \neq F^+$$

      - für alle Attribute $Y \in B$ gilt:

        $$(F_c - \{A \rightarrow B\} \cup \{A \rightarrow (B-Y)\})^+ \neq F^+$$

    - Jede linke Seite der FDs in F_c kommt nur einmal vor.

Algorithmus

- Gegeben: Menge F mit funktionalen Abhängigkeiten
  1. Führe für jede FD $A \rightarrow B$ aus F die ***Linksreduktion*** *durch:*

     Überprüfe für alle $X \in A$, ob X überflüssig ist, d.h. ob
     $B \subseteq$ Hülle(F, A – X).

     Ist dies der Fall, ersetze in F $A \rightarrow B$ durch $(A-X) \rightarrow B$.

  2. Führe für jede verbliebene FD $A \rightarrow B$ die **Rechtsreduktion** durch: Überprüfe für alle $Y \in B$, ob Y überflüssig ist, d. h. ob

     $Y \in$ Hülle$(F – \{A \rightarrow B\} \cup \{A \rightarrow (B-Y)\}, A)$.

     Ist dies der Fall, wird $A \rightarrow B$ durch $A \rightarrow (B-Y)$ ersetzt.

  3. Entferne die FDs der Form $A \rightarrow \varnothing$ (die im 2-ten Schritt entstanden sind)

  4. Ersetze alle FDs der Form $A \rightarrow B_1, \ldots, A \rightarrow B_k$ durch

     $A \rightarrow B_1 \cup \ldots \cup B_k$

Beispiel

- Menge F = {A→ B, B → C , A ∪ B → C }
- Schritt 1:
- Schritt 2:
- Schritt 3:
- Schritt 4:

Zerlegung einer Relation

- Um Anomalien zu beseitigen, wird eine Relation r mit Schema RS in n Relationen $r_1,\ldots,r_n$ mit Schemata RS_1, …, RS_n zerlegt.
  - Folgende Eigenschaften sollen dabei erfüllt sein:
    - **Kein Informationsverlust**, d.h. Relation r muss aus den Relationen $r_1,\ldots,r_n$ wieder rekonstruierbar sein.

$$r = \pi_{RS_1}(r) \,|x|\, \ldots \,|x|\, \pi_{RS_n}(r)$$

    - Alle FDs, die für die Relation r gelten, sollen für $r_1,\ldots,r_n$ effizient überprüfbar bleiben. (→ **lokale Erhaltung** der FDs)

Theorem

- Sei RS ein Schema und F_{RS} die Menge der FDs. Eine Zerlegung von RS in Schemata RS_1 und RS_2 hat keinen Informationsverlust, falls eine der folgenden Bedingungen gilt:
  - $(RS_1 \cap RS_2) \rightarrow RS_1 \in F_{RS}^+$
  - $(RS_1 \cap RS_2) \rightarrow RS_2 \in F_{RS}^+$

Anwendung des Theorems

- **Beispiel:**
  - Ist die Zerlegung der Relation LieferantW(LName, LAdr, Ware, Preis) in die Relationen
    - Lieferant(LName, LAdr)
    - Angebot(LName, Ware, Preis)

    verlustlos?

  - Ist die Zerlegung
    - Lieferant(LName, LAdr, Ware)
    - Angebot(Ware, Preis)

    verlustlos?
    - Wir können hierzu keine Aussage treffen, da das Theorem nur eine hinreichende, aber **keine notwendige** Bedingung!
    - Gibt es noch ein besseres Theorem?

Hüllentreue: Lokale Erhaltung von FDs

- Lokale Erhaltung funktionaler Abhängigkeiten
  - Wunsch: Menge F von FDs, die für die Relation r gelten, sollen *lokal auf den zerlegten Relationen $r_1,\ldots,r_n$ überprüfbar sein.*

- Formal kann dies wie folgt ausgedrückt werden. Seien $F_1, \ldots, F_n$ die lokalen Mengen von FDs für die Relationen $r_1, \ldots, r_n$. Dann soll Folgendes gelten:

$$F^+ = (F_1 \cup \ldots \cup F_n)^+$$

  Dann wird die Zerlegung als **hüllentreu** bezeichnet.

Beispiel

- **Beispiel aus Kemper/Eickler**
  - Sei die Relation PV(Straße, Ort, BLand, PLZ) gegeben Es sollen folgende Bedingungen gelten:
    - Orte werden durch “Ort” und “BLand” eindeutig charakterisiert.
    - Innerhalb einer Straße ändert sich “PLZ” nicht.
    - PLZ-Gebiete gehen nicht über Ortsgrenzen, Orte nicht über Bundeslandgrenzen.
  - FDs
    - {PLZ} → {Ort, BLand}
    - {Straße, Ort, BLand} → {PLZ}
  - Welche Eigenschaften besitzt die Zerlegung {PLZ, Straße} und {PLZ, Ort, BLand}?

4.3.2 Normalformen

- Durch Normalformen wird definiert, was unter einem „guten“ Datenbankdesign zu verstehen ist.
  - Vermeidung von Redundanz und Anomalien

- Grundlage von einer Klasse von Normalformen
  - **Funktionale Abhängigkeiten**

  - **Schlüsselkandidaten**
    - Besitzt eine Relation mehr als einen Schlüssel, so spricht man auch von Schlüsselkandidaten.

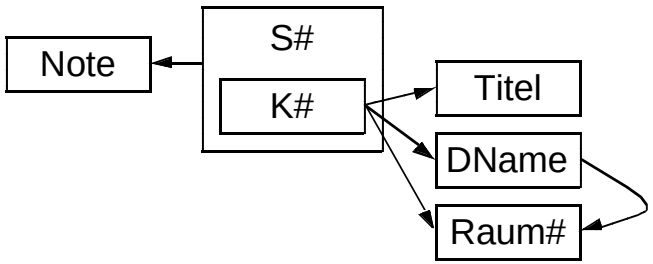
  - **Prime-Attribut**
    - ist ein Attribut, das Teil eines Schlüsselkandidaten ist.
    - **Nicht-Prime** Attribute sind dann genau die anderen Attribute.

Erste Normalform

- **Def. (1. Normalform)**
  - Ein Relationenschema ist in der *1. Normalform (1NF), wenn alle Attribute nur atomare Werte, die nicht weiter zerlegbar sind, annehmen können.*
    - *Wertebereiche wie z. B. STRING, INTEGER etc.*
    - *Mengen von Werten oder sonstige Strukturen (z. B.Tupel) sind für Attributwerte nicht erlaubt.*
- **Anmerkungen**
  - 1. Normalform wurde bereits bei der Definition einer Relation gefordert.
  - Heutige relationale Systeme bieten inzwischen Möglichkeiten, um Relationen in Non-First-Normal-Form (NF^2) zu erzeugen.
    - PostgreSQL unterstützt z. B. Arrays

Zweite Normalform

- **Def. (2. Normalform)**
  - Ein Relationenschema ist in der *2. Normalform (2NF), wenn jedes Nicht-Prime Attribut von jedem Schlüsselkandidat* ***voll*** *funktional abhängig ist.*
- ***Beispiel***
  - Leistungsnachweis (S#, K#, Titel, DName, Raum#, Note)
  - Tupel ***(s, k, t, d, r, n)*** *bedeutet:* Student ***s*** hat die Note ***n*** erzielt im Kurs mit Nummer ***k***, der den Titel ***t*** trug und im Raum mit Nummer ***r*** vom Dozenten ***d*** abgehalten wurde.
  - Folgende Abhängigkeiten bestehen
    - {S#, K#} → {Note}
    - {K#} → {Titel}
    - {K#} → {DName}
    - {DName} → {Raum#}
    - {K#} → {Raum#}



Transformation in 2NF

Warum?

- Relationenschema ‘Leistungsnachweis’ ist nicht in 2NF. Folgende Anomalien können auftreten:
  - Informationen über einen neuen Kurs sind nur dann verfügbar, wenn bereits ein Student für diesen Kurs eingetragen ist.
  - Dozent ist nur dann in der Datenbank, wenn sie einen Kurs hält
  - Namensänderung eines Kurses ist sehr aufwendig.
  - Falls alle Studenten einen Kurs 27 verlassen (➔ Löschen der Tupel), verschwinden alle Informationen über den Kurs.

- Transformation in 2NF behebt diese Anomalien:
  - Aufspalten der Relation ‘Leistungsnachweis’ in folgende zwei Relationen (in 2NF):
    - Leistungsnachweis (S#, K#, Note)
    - Kurs (K#, Titel, DName, Raum#)

Allgemein

- **Beobachtung**
  - 2NF kann nur dann verletzt werden, wenn Schlüsselkandidaten zusammengesetzt sind.
- **Transformation**
  - **Partielle** funktionale Abhängigkeiten nicht-primer Attribute von einem Schlüsselkandidat werden beseitigt.
    - Erzeugung eines entsprechenden Relationenschemas
- **Bedeutung der 2NF**
  - Gering, da noch eine weitere Verschärfung notwendig ist.

Dritte Normalform

- **Def. (3. Normalform)**
  - Ein Relationenschema RS ist in *3. Normalform (3NF), wenn es **kein** Attribut A in RS mit folgenden zwei Eigenschaften gibt:*
    - *A ist Nicht-Prime*

    und

    - *A ist von einem Schlüsselkandidat transitiv abhängig.*

- **Beobachtung**
  - 3NF beseitigt Abhängigkeiten von Nicht-Prime Attributen.
  - Jede Relation in 3NF ist auch in 2NF.

Transformation in 3NF

- **Problem**
  - Relationenschema ‘Kurs’ ist nicht in 3NF, da die Abhängigkeit DName → Raum# besteht und DName weder Schlüssel noch Raum# prim ist.
  - Folgende Anomalien können auftreten
    - Informationen über Dozenten und Raum sind ohne Zuordnung eines Kurses nicht verfügbar
    - Ändern der Raumnummer eines Dozenten bedingt die Änderung für jeden Kurs
    - Falls ein Dozent keinen Kurs gibt, werden alle Informationen über den Dozent und seinen Raum aus der Datenbank gelöscht.
- **Schema in 3NF durch Aufteilen der Relation**
  - Leistungsnachweis (S#, K#, Note)
  - Kurs (K#, Titel, DName)
  - Dozent (DName, Raum#)

Synthesealgorithmus

- **Ziel ist die Zerlegung einer Universalrelation r mit funktionalen Abhängigkeiten F in Relationen $r_1,\ldots,r_n$ mit folgenden Bedingungen:**
  - kein Informationsverlust,
  - Bewahrung der funktionalen Abhängigkeiten,
  - Relationen $r_1,\ldots,r_n$ erfüllen die dritte Normalform.
- **Ansatz zur Erzeugung einer Datenbank in 3NF**
  - Top-Down (Dekomposition)
    - Ausgehend von einer Universalrelation werden Relationen, die nicht in 3NF, rekursiv zerlegt.
  - Bottom-Up (Synthese)
    - Ausgehend von der Menge der FDs werden 3NF-Relationen erzeugt, die dann rekursive miteinander verschmolzen werden (solange diese in 3NF bleiben).

Synthesealgorithmus

Eingabe:

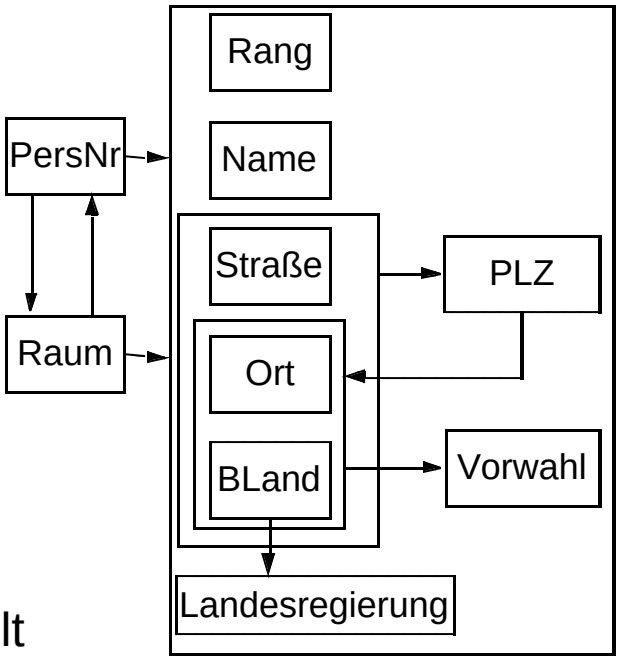
- Menge U der Attribute, F = Menge von FDs

Ausgabe: Relationenschemata in 3 NF

1. Bestimme kanonische Überdeckung F_c der Menge F.
2. Führe für jede $A \rightarrow B \in F_c$ folgende Anweisung aus:
   - Erzeuge ein Relationenschema $RS_A = A \cup B$ und ordne RS alle FDs $F_A = \{C \rightarrow D \mid C \cup D \subseteq RS_A\}$ zu.
3. Falls keines der in Schritt 2 erzeugten Schemata einen Kandidatenschlüssel K der Menge U enthält, wird zusätzlich eine Relation mit dem Schema $RS_K = K$ und $F_K = \emptyset$ erzeugt (K = Schlüssel von U).
4. Eliminiere die Schemata, die in einem anderen Schema enthalten sind.

Beispiel

- **Beispiel:**
  - U = ProfessorenAdr = {PersNr, Raum, Rang, Name, Straße, Ort, BLand, Landesreg., PLZ, Vorwahl}
  - Annahmen
    - Ort ist der Erstwohnsitz des Profs
    - Landesregierung ist die Partei der Ministerpräsidentin
    - Ortsnamen sind eindeutig innerhalb der Bundesländer
    - PLZ ändert sich nicht innerhalb einer Straße
    - Städte und Straßen liegen vollständig in Bundesländern
    - ein Prof hat genau ein Büro (und er teilt es nicht)
    - …



Beispiel (2)

- **Schritt 1: kanonische Überdeckung**
  - FD_1 {PersNr} → {Raum, Name, Rang, Straße, Ort, BLand}
  - FD_2 {Raum} → {PersNr}
  - FD_3 {Sraße, Ort, BLand} → {PLZ}
  - FD_4 {Ort, BLand} → {Vorwahl}
  - FD_5 {BLand} → {Landesregierung}
  - FD_6 {PLZ} → {Ort, BLand}
- **Schritt 2:**
  - Verarbeitung von FD_1
    - {PersNr, Name, Rang, Raum, Straße, Ort, Bland}
    - FD_1 und FD_2 werden zugeordnet
  - Verarbeitung von FD_3
    - {Straße, Ort, BLand, PLZ}
    - FD_3 und FD_6 werden zugeordnet

Beispiel (3)

- Verarbeitung von FD_4
  - {Ort, BLand, Vorwahl}
  - FD_4 wird zugeordnet
- Verarbeitung von FD_5
  - {BLand, Landesregierung}
  - FD_5 wird zugeordnet

- **Schritt 3:**
  - {PersNr} ist Kandidatenschlüssel des ursprünglichen Schemas und befindet sich in dem ersten Relationenschema.

- **Schritt 4:**
  - Bedingung ist für keines der vier Relationenschemata erfüllt. ➔ Nichts mehr zu tun!

Wichtiges Resultat

- **Zu einer Menge U von Attributen und einer Menge von FDs, liefert der Synthesealgorithmus ein Datenbankschema mit folgenden Eigenschaften:**
  1. Alle Relationen sind in 3NF
     - Es gilt sogar, dass die Anzahl der Relationen minimal ist.
  2. Hüllentreu
     - Lokale Überprüfung aller FDs möglich
  3. Kein Informationsverlust
     - Rekonstruktion der Universalrelation möglich

Boyce-Codd Normalform

- **Def. (Boyce/Codd-NF)**
  - Ein Relationenschema RS ist in ***Boyce/Codd-Normalform (BCNF),*** *wenn für alle funktionalen Abhängigkeiten* $A \rightarrow \{X\}$ *mit* $A \subseteq RS$, $X \in RS - A$ *gilt:*
    - A enthält einen Schlüsselkandidaten von RS.

- **Beobachtung**
  - Die Boyce/Codd-Normalform beseitigt funktionale Abhängigkeiten unter Attributen, die prim sind.
  - Jede Relation in BCNF ist auch in in 3NF.

Beispiel

- *Autoverzeichnis (Hersteller, HerstellerNr, ModellNr)*
  - Folgende funktionale Abhängigkeiten bestehen:
    - Hersteller → HerstellerNr

      1:1-Beziehung zwischen Hersteller und HerstellerNr
    - HerstellerNr → Hersteller
  - Beispiel ist in 3NF (da alle Attribute sind prim), aber nicht in BCNF.
  - Folgende Anomalien können auftreten:
    - Einfügen des selben Herstellers mit verschiedenen HerstellerNr ist möglich.
    - 1:1-Beziehung von Hersteller und HerstellerNr ist an die ModellNr gekoppelt.

Wichtiges Resultat

- **Zu einer Menge U von Attributen und einer Menge von FDs, gibt es ein Datenbankschema mit folgenden Eigenschaften:**
  1. Alle Relationen sind in BCNF
     - Es gilt sogar, dass die Anzahl der Relationen minimal ist.
  2. Kein Informationsverlust
     - Rekonstruktion der Universalrelation möglich
- **Schlechte Nachricht**
  - Es kann nicht immer eine hüllentreue Zerlegung gefunden werden.

  ➔ Man gibt sich mit 3NF in der Praxis zufrieden.

4.3.3 Normalformen und mehrwertige Abhängigkeiten

- Nicht alle semantischen Beziehungen unter Attributen können durch FD adäquat modelliert werden.

Mehrwertige Abhängigkeiten

- Verallgemeinerung funktionaler Abhängigkeiten
- Beispiel
  - Relation Buch mit Schema {ISBN, Autor, Stichwort}

ISBN	Autor	Stichwort
3486598341	Kemper	Normalformen
3486598341	Eickler	Normalformen
3486598341	Kemper	Relationale Algebra
3486598341	Eickler	Relationale Algebra

  - Ein Buch kann mehrere Autoren und mehrere Stichwörter besitzen

    ➔ Autor bzw. Stichwort sind mehrwertig abhängig von ISBN

Mehrwertige Abhängigkeiten

- **Def. (mehrwertig abhängig):**
  - Sei RS ein Relationschema und A, B $\subseteq$ RS und C = RS - A $\cup$B . Dann ist B ***mehrwertig abhängig von A, A →→ B,*** wenn für alle Relationen r $\in$REL(RS) gilt:
    - Für jedes Paar von Tupel t_1, $t_2 \in$ r mit $t_1[A] = t_2[A]$ existieren zwei Tupel t_3, $t_4 \in$ r mit $t_3[A] = t_4[A] = t_1[A]$ mit
      $t_3[B] = t_1[B]$ $\quad$ $t_4[B] = t_2[B]$
      $t_3[C] = t_2[C]$ $\quad$ $t_4[C] = t_1[C]$
  - Es wird die Kurzschreibweise **MVD** (**multi-value dependency**) für eine mehrwertige Abhängigkeit benutzt.

Beispiel

- In unserem Beispiel mit der Relation Buch existieren folgende mehrwertige Abhängigkeiten:
  - {ISBN} →→ {Autor}
  - {ISBN} →→ {Stichwort}
- Offensichtlich führt dies zu Redundanzen, die möglichst vermieden werden sollen.
- Idee (Dekomposition)
  - Zerlegen des Relationschemas Buch in zwei Schemata RS_1 und RS_2
    - RS_1 = {ISBN, Autor}
    - RS_2 = {ISBN, Stichwort}
  - Es gilt sogar: Die Zerlegung ist verlustfrei!

Das Spiel beginnt von vorne

- **Analog zu FDs kann man nun für MVDs vorgehen.**
  - Zu einer Menge von M von MVDs kann man mit Hilfe von Regeln die **Hülle M^+** von M berechnet werden.
  - Zu der Hülle M^+ kann dann die **kanonische Überdeckung** berechnet werden, aus der M+ generiert werden kann.

- **… aber es gibt noch ein wichtiges Resultat!**

Theorem

Sei RS eine Relation und M_{RS} die Menge der MVDs. Eine Zerlegung von RS in Schemata RS_1 und RS_2 hat keinen Informationsverlust genau dann, falls mindestens eine der folgenden Bedingungen gilt:

- $RS1 \twoheadrightarrow (RS1 \cap RS2) \in M_{RS}^+$
- $RS2 \twoheadrightarrow (RS1 \cap RS2) \in M_{RS}^+$

Vierte Normalform

- Vierte Normalform ist eine Verstärkung der BCNF
  - Vermeidung der durch mehrwertige Abhängigkeiten verursachten Redundanz
- Sei *RS* ein Relationenschema und $A, B \subseteq RS$. Eine mehrwertige Abhängigkeit $A \twoheadrightarrow B$ ist ***trivial***, *falls eine der folgenden Bedingungen gilt:*
  - $B \subseteq A$
  - $B = RS - A$

- **Definition (4NF):**
  - Sei *RS* ein Relationschema und *M* die zugehörige Menge mehrwertiger Abhängigkeiten. RS ist in vierter Normalform (4NF), wenn für jede nicht-triviale mehrwertige Abhängigkeit $A \twoheadrightarrow B \in M^+$ folgende Bedingung gilt:
    - *A* enthält einen Schlüsselkandidaten von *RS*.

4.4 Grenzen der Normalformen

- **Sinn und Zweck von Normalformen**
  - Unterstützung von Lastprofilen mit **vielen Änderungen** in der Datenbank.
    - Klassische Kontoverwaltung bei Banken
  - Vermeidung von Redundanz

    ➔ Verbesserung der Performance

- **Es gibt auch viele Datenbankanwendungen, die**
  - keine (direkten) Änderungsoperationen erlauben und „nur“ Lesen der Daten unterstützen.
  - In solchen Anwendungen sollte man sich genau überlegen, ob Relationen in Normalform sein sollten.

Datawarehousing

- **Definition (**siehe Building the Data Warehouse, Immon**)**
  - Ein Data-Warehouse ist eine **themenorientierte**, historische und autonome Datenbank eines Unternehmens, in der Daten aus verschiedenen unabhängigen **heterogenen Quellsystemen** integriert und verwaltet werden. Ziel ist, einem Unternehmen durch zeitbezogene **Abfragen und Analysen** entscheidungsunterstützende Ergebnisse zu liefern.

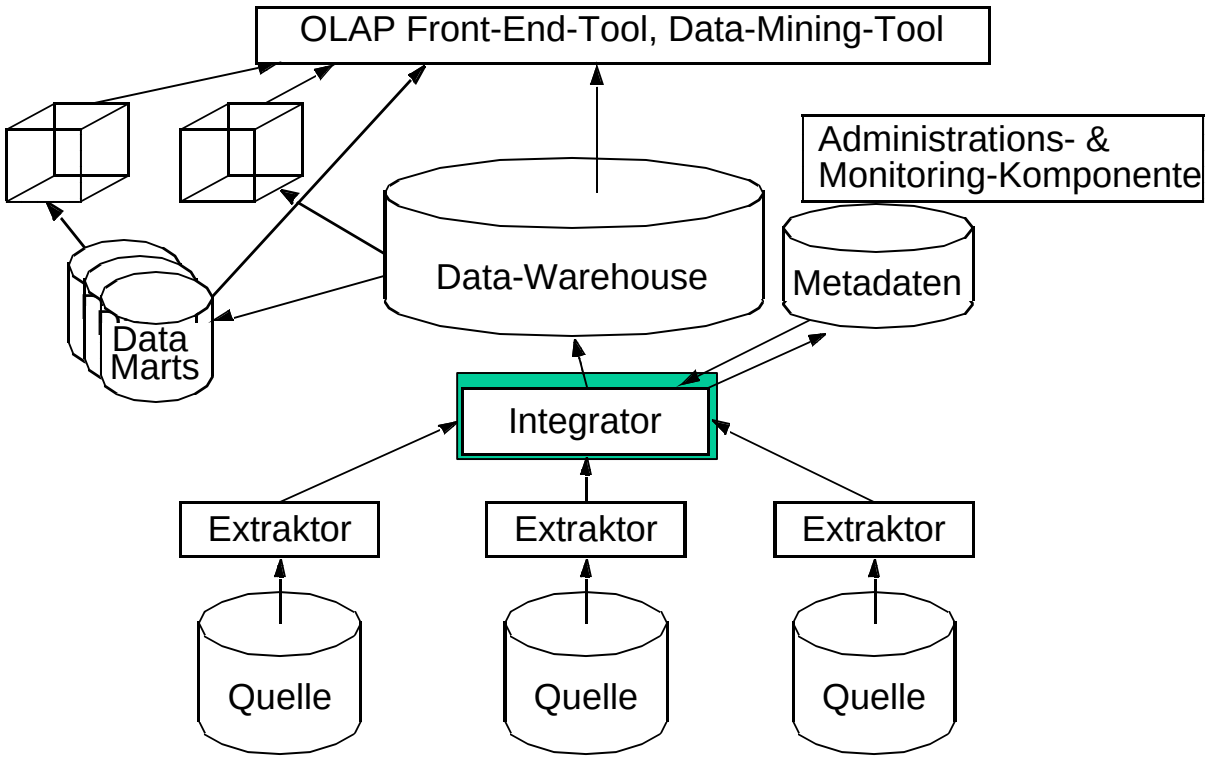
- **Themenorientierung**
  - Entscheidungsrelevante Sachthemen eines Unternehmens stehen im Vordergrund
    - umsatzstärkste **Kunden**
    - kostenintensivste **Produkte**

    Man spricht dann auch von einer **multidimensionalen Datenbank.**

Integration

- **Integration**
  - Verknüpfung der Daten aus heterogenen Datenbanken (Quellsystemen)
  - **Aktiver Ansatz (eager, in advance)**
    - Extraktion aller Daten aus den Quellsystemen im voraus
      - ➔ kein Zugriff auf Quellsysteme bei Anfrageausführung
    - **Keine** Änderungen im Data-Warehouse durch den Endbenutzer

Architektur

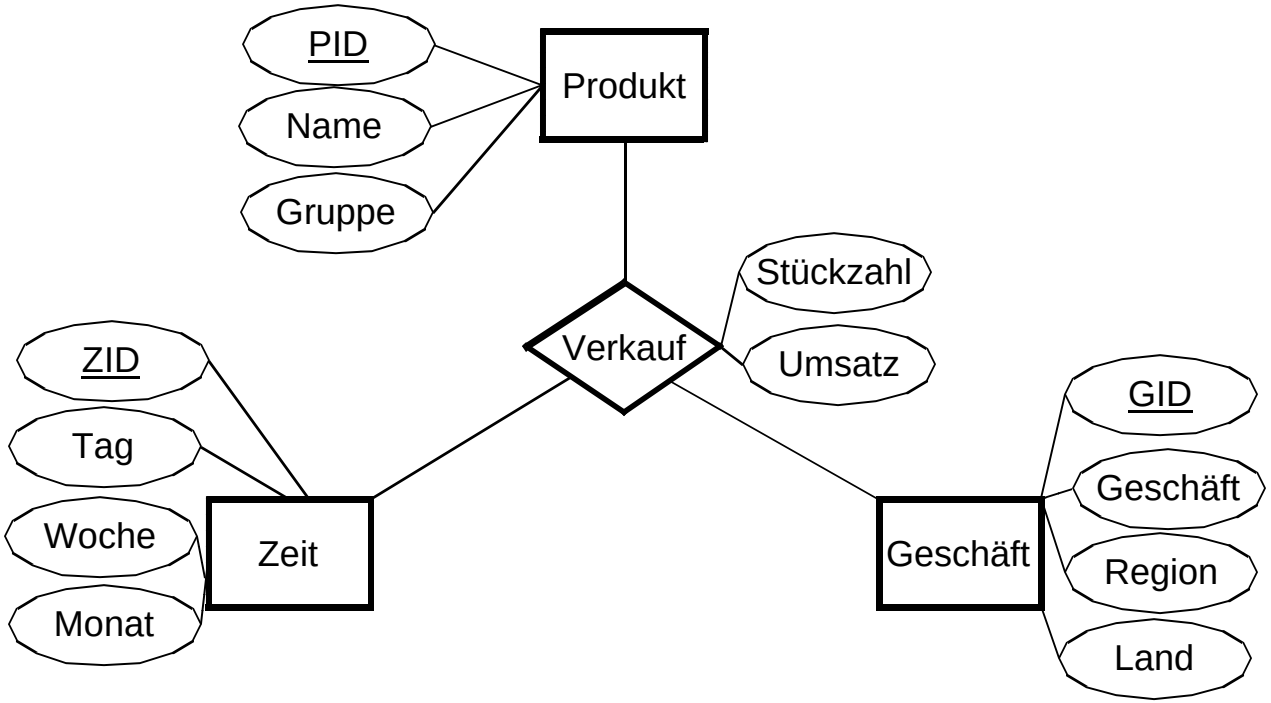


Mehrdimensionales Datenmodell

- Datawarehouse basiert auf einem mehrdimensionalen Datenmodell (konzeptionelle Ebene)
- Bestandteile des Datenmodells
  - **Messgrößen (Fakten)**
    - numerische Wertebereiche
    - Beispiele: Umsatz, verkaufte Einheiten, Kosten, …
  - **Dimensionen**
    - Messgrößen liegen in einem mehrdimensionalen Kontext
      - Beispiel: Umsatz hängt z. B. ab von den Dimensionen **Produkt**, **Ort** und **Zeit**.

        ➔ "**natürliche Parameter"** des Umsatzes.
      - Jede Dimension setzt sich i. A. zusammen aus mehreren Attributen.
        - Z. B.: Ort besitzt als Attribute Region, Land und Geschäft.
        - Es gelten FDs zwischen diesen Attributen!

Schema als einfaches ER-Modell



Umsetzung als Star-Schema

- Ein **Star-Schema** besteht aus einer Menge von Relationen $D_1,\ldots,D_n$, F mit folgenden Eigenschaften:
  - Jede Relation D_i modelliert eine Dimension. Zusätzlich zu den Attributen der Dimension bekommt D_i einen künstlichen Primärschlüssel d_i zugeordnet. D_i wird dann auch als **Dimensionsrelation** bezeichnet**.**
  - Die Relation F verbindet die n Dimensionen miteinander, indem die Fremdschlüssel der Dimensionsrelation (d. h. $d_1,\ldots,d_n$) und die entsprechenden Messgrößen als Attribute umgesetzt werden. Diese Relation wird dann auch als **Faktenrelation** bezeichnet**.**

- **Beobachtung**
  - Das Star-Schema (genauer die Dimensionsrelationen) liegen nicht in dritter Normalform vor.

Analytische Anfragen

- Finde die Quartalsumsätze aller Geschäfte differenziert nach Ländern im Jahr 2012 für die Produktgruppe Kaffee.

```
select   PName, Land, Quartal, sum(Umsatz)
from     Produkt P, Geschäft G, Zeit Z, FaktenRel F
where    Jahr = 2012                          and
         Produktgruppe = “Kaffee”             and
         Z.ZID = F.ZID                        and
         P.PID = F.PID                        and
         G.GID = F.GID
group by PName, Land, Quartal;
```

Bemerkungen

- Die spezielle Form des Joins (Anfrage) wird auch als **Star-Join (Star-Anfrage)** bezeichnet.
- In einigen Systemen gibt es spezielle SQL-Syntax zur Unterstützung von solchen analytischen Anfragen.

Eisberg-Anfrage

- Finde die10 stärksten Quartalsumsätze aller Geschäfte differenziert nach Ländern im Jahr 2012 für die Produktgruppe Kaffee.

```
select   PName, Land, Quartal, sum(Umsatz) as q_umsatz
from     Produkt P, Geschäft G, Zeit Z, FaktenRel F
where    Jahr = 2006 and
         Produktgruppe = “Kaffee”   and
         Z.ZID = F.ZID              and
         P.PID = F.PID              and
         G.GID = F.GID
group by PName, Land, Quartal
order by q_umsatz
limits   10;
```

- **Bemerkungen**
  - Dies ist auch eine typische Anfrage im Bereich Data Warehouse.

Zusammenfassung

- **Zentrale Frage in diesem Kapitel**
  - Modellierung von Daten
- **ER-Modellierung**
  - Einfaches Werkzeug zur konzeptuellen Modellierung
- **Normalformtheorie (mit viel Anwendungsbezug)**
  - Funktionale und mehrwertige Abhängigkeiten
  - Normalformen
    - 1NF, 2NF, 3NF, BCNF, 4NF
  - Synthesealgorithmus
- **Modellierung leseintensiver Datenbanken**
  - Datawarehouse
  - Mehrdimensionales Datenmodell