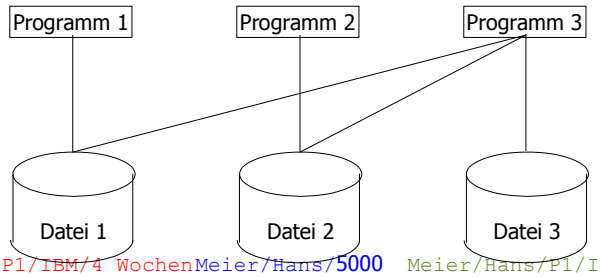
1.Grundkonzepte für die Architektur von Datenbanksystemen (DaBa)

1.1Ausgangssituation

Frühere Anwendungssysteme sind gekennzeichnet, dass zusammengehörige Programme auf vielen Dateien operieren.

Beispiel: Verwalten von Projekten einer Firma

1. Es gibt 3 Dateien
   - *Datei 1:* - Daten aller laufenden, aller neuen und aller abgeschlossenen Projekte
   - *Datei 2:* - Daten aller Mitarbeiter
   - *Datei 3:* - Daten, die belegen welches Projekt von welchem Angestelleten betreut wird
2. Es gibt 3 Programme
   - *Programm 1* - Eintragen neuer Projekte (Zugriff auf Datei 1)
   - *Programm 2* - Eintragen neuer Mitarbeiter (Zugriff auf Datei 2)
   - *Programm 3* - Verwalten der Projekte (Zugriff auf alle Dateien)



P1/IBM/4 WochenMeier/Hans/5000 Meier/Hans/P1/IBM/4 Wochen

Bsp. Datensatz

Nachteile:

- *hohe Redundanz :*
  - viele Dateien enthalten Informationen, die in anderen schon enthalten sind
    ➔ hohe Speicherkapazität
    ➔ hoher Aufwand bei der Aktualisierung der Daten
- *Probleme der Inkonsitenz:*
  - Bei der Aktualisierung können Fehler aufttreten, da u.U. nicht in allen Dateien geändert wird
- *hoher Grad der Datenabhängigkeit*
  - Logische Datenabhängigkeit:
    – Jedes Programm muss wissen, wie die Daten aufgebaut sind. Ändert sich die Struktur, so müssen u.U. die darauf zugreifenden Programme geändert werden.
    z.B. Aufbau von Datei 2 ist neu: Meier/meier@t-online.de/Hans/5000
    → Programm 3 muss geändert werden.
  - Physische Datenabhängigkeit:
    – Programme müssen wissen, wo sich die Dateien auf der Festplatte befinden. Ändert sich z.B. der Ort, so muss auch das Programm geändert werden

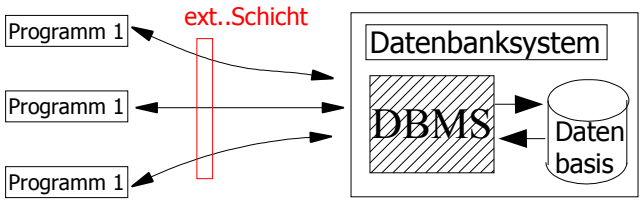
  ⇒ **hoher Wartungsaufwand**
- *Datenschutz und Datensicherheit*
  - Anwender können auf Daten zugreifen, schutzwürdig sind (z.B. Gehalt in Datei 2 im Bsp.). Anwender können unberechtigterweise Daten verändern (z.B. Gehalt erhöhen).

1.2 Konzeption von DABA

- *Anforderung:*
  - Vermeidung der logischen und physischen Datenabhängigkeit
  - Vermeidung des unkontrollierten Zugriffs auf die Daten

Es werden zwei Elemente benötigt:

- *Datenbasis*
  - Menge aller Daten, die von den Programmen benötigt werden
  - die Daten werden nach einheitlichen Regeln abgelegt
- *Datenbankmanagementsystem (DBMS)*
  - Kontrollprogramm, über das der Zugriff auf die Daten möglich ist
- *es existiert eine Schnittstelle, über die die Programme dem DBMS Aufträge geben.*



SQL structure query language

Jedes Programm hat auf die Daten seine eigene Sicht. Das bedeutet, nur die für das Programm notwendigen Daten sind zu greifen.

Beispiel:
Ein Programm, das sie Projektdaten verwaltet, kann nicht auf die Mitarbeiterdaten zugreifen. Dies muss dem DBMS mitgeteilt werden (→**view** einrichten).

1.3Vorteile der Organisation der Daten in Datenbanken

- *Verminderung der Redundanz*
  - Daten sind nach einheitlichen Regeln abgelegt.
  - Falls Redundanz notwendig, erfolgt diese unter Kontrolle des DBMS

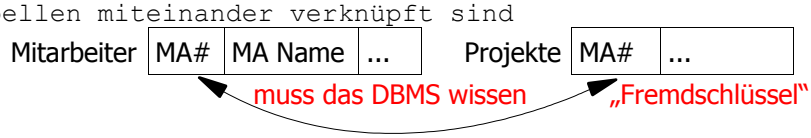
MA#	Name	...	...
1	Meier	Hans	5000

MA#	Projekt#
1	1

Kontrolle

Logische Datenunabhängigkeit
- Da jedes Programm seine spezifische Sicht hat, muss es nicht geändert werden,wenn z.B. eine Spalte in der Datenbasis hinzugefügt wird (z.B. E-Mail Adresse).

- *Physische Datenunabhängigkeit*
  - Programme sind unabhängig von der internen Datenorganisation
- *Gewährleistung von Datenkonsistenz*
  - Semantische Integrität:
    - ➔ DBMS muss wissen, dass z.B. in einer Spalte keine negativen Werte vorkommen dürfen, oder dass für eine Spalte auf jeden Fall ein Wert angegeben sein muss.
  - Operationale Integrität
    - ➔ Das DBMS muss wissen, wie Daten aus verschiedenen Tabellen miteinander verknüpft sind



- *Gewährleistung von Datenschutz*
  - ➔ Da jedes Programm seine externe Sicht hat, kann es nur auf Daten zugreifen, die der Sicht zugeordnet sind.

1.4 Das 3-Ebenen Architektur-Konzept

- *wichtiger Schritt für den Entwurf der Datenbasis*
- *1975 entwickelt von ANSI-SPARC-Komitee*
- *Idee*
  - Beschreibung der Daten auf 3 verschiedenen Ebenen
  - auf jeder Ebene hat man eine andere Sichtweise
  - vorgeschlagene Ebenen
    - ➔ externe Ebene
    - ➔ konzeptionelle Ebene
    - ➔ interne Ebene

Konzeptionelle Ebene

- beschreibt Gesamtschau der Daten
- Daten werden logisch beschrieben, d.h. unabhängig von Gesichtspunkten der Datenverarbeitung
- wird mit Hilfe der **Data Descritption Language (DDL)** beschrieben. DDL gehört zur SQL
- Ergebnis: *konzeptionelles Schema*

Externe Ebene

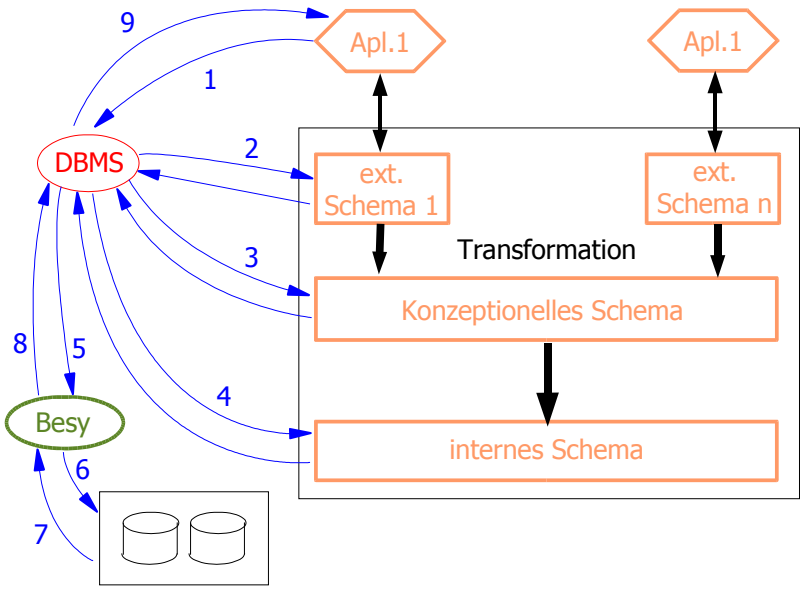
- individuelle Sicht auf die auf die Daten
- Daten werden in einem eigenen externen Modell dargestellt
- Ergebnis: *externes Schema*

Interne Ebene

- physische Datenorganisation, u.a. Angabe über den Aufbau der gespeicherten Datensätze, Zugriff darauf
- in dieser Ebene muss die Art und Weise, wie das gewählte DBMS die Daten ablegen möchte, berücksichtigt werden
- Ergebnis: *internes Schema*

Die drei Schemata werden zu den eigentlichen Daten (z.B. die MA-Daten) in der Datenbank abgelegt. Man bezeichnet diese auch als Metadaten, da sie Informationen über die Daten enthalten. In der Datenbankwelt werden die Metadaten auch als DATA DICTIONARY bezeichnet. Auf diese greift das DBMS zu, um entscheiden zu können, ob der Zugriff erlaubt ist.

Verwalten der Ebenen durch das DBMS



1. Befehlsübergabe
2. Anfrage, ob Zugriff zulässig
3. Anfrage, wie ist der Zugriff in konz.Schema realisiert
4. Wie ist der phys. Zugriff
5. bes bekommt den Auftrag, Daten von der HD zu holen oder zu schreiben
6. bes übergibt den Auftrag an die Festplatte
7. Festplatte liefert Daten an Betriebssystem
8. bes liefert Daten an DBMS
9. DBMS übergibt Daten an Anwendung, es muss vorher aber das dazu geh. ext. Schema berücksichtigen

2.Datenbankentwurf

2.1Definition „Datenbankentwurf"

Datenbankentwurf legt die logische und physische Struktur der Datenbasis fest. Die Festlegung
erfolgt mit dem Ziel, die Informationsbedürfnisse der Anwender zu berücksichtigen

2.2Datenbank-Anomalie

Vorlesung-Professoren

Prof.#	Name Prof.	Raum#	Vorl.#	Vorl. Name	Sem.Wo.Std
212	Rösch	1-215	3	ENDA	2
212	Rösch	1-215	4	DABA	4
		... ...			
225	Lang	1-234	9	DATÜ	3
226	Rausch	1-233	2	IGRU	6
		... ...			
?	dummy	16-17	10	UNIX	4

- *Trage Vorlesung UNIX ein, die noch keinem Professor zugeordnet ist*
  - Führt man für die nicht bekannten Werte Dummywerte ein, muss jede Anwendung wissen, was die Werte bedeuten. Man spricht von einer INSERT-ANOMALIE
- *Lösche Vorlesung DATÜ. Es ist die einzige Vorlesung, die LANG hält.*
  - Durch das Löschen wird in diesem Fall auch die Information über LANG gelöscht.
    Man spricht von einer DELETE-ANOMALIE
- *Ändere Raum# bei RÖSCH*
  - Informationen existieren mehrmals, d.h. an vielen Stellen muss die Raum-Nr. geändert werden. Falls dies nicht gemacht wird, kommt es zu einer Dateninkonsitenz. Auch ist die Änderung unnötig aufwendig. Man spricht von einer UPDATE-ANOMALIE

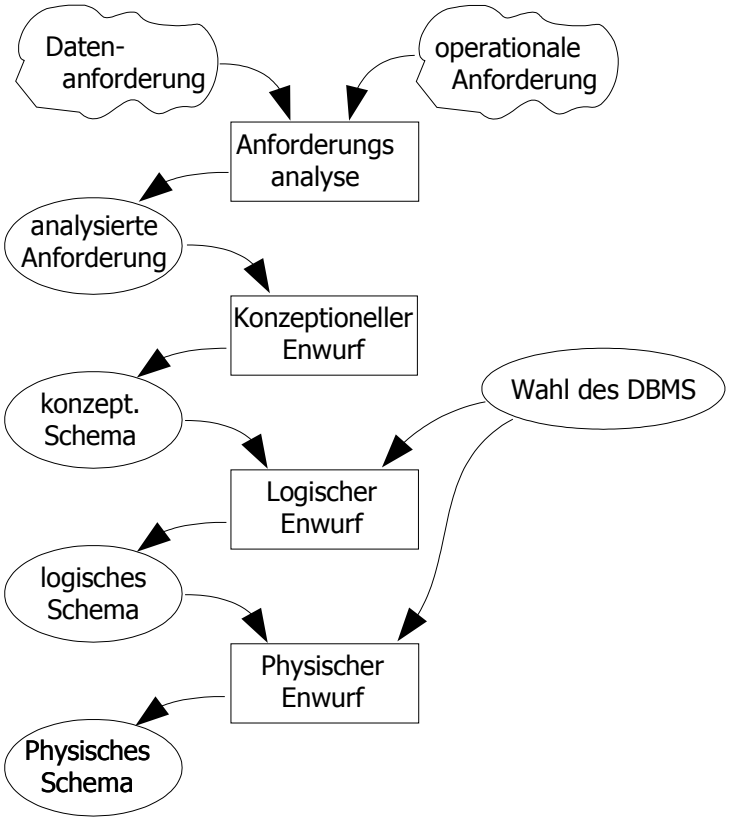
Es ist eine Methode notwendig, mit der das Auftreten der Anomalien verhindert wird.

2.3Qualitätskriterien

Was muss die Methode erfüllen?

1. Vollständigkeit
   - Alle Aspekte müssen erfasst sein, die für die Datenbasis relevant sind.
2. Korrektheit
   - Das zugrunde liegende Datenmodell muss korrekt angewendet worden sein
3. Minimalität
   - Jeder Aspekt kommt nur einmal vor. Muss ein Aspekt mehrmals aufgeführt werden (Redundanz), muss dies dem DBMS mitgeteilt werden
4. Lesbarkeit
   - Diagrammtechnik, vernünftige Namenswahl, gute Dokumentation
5. Normalität
   - Falls das relationale Datenmodell verwendet wird, muss sichergestellt sein, dass die Relationen normalisiert sind. (→siehe später)

2.4Entwurfsphasen



- *Anforderungsanalyse*
  - statische Information
    - ➔ welche Information
    - ➔ Beschreibung der Objekte
  - dynamische Aktivitäten
    - ➔ welche Operationen
    - ➔ wie häufig
    - ➔ welches Datenvolumen
  - Benutzerkreis
    - ➔ wer arbeitet damit
    - ➔ welche Rechte
    - ➔ welche Aufgaben
- *konzeptioneller Entwurf*
  - ➔ Abstraktion von der realen Miniwelt
    - **Entity-Relationship-Modell**
- *logischer Entwurf*
  - ➔ Umsetzung des konzeptionellen Schemas in die Sprache des Zieldatenmodells
    - relationales Datenmodell
- *physischer Entwurf*
  - ➔ Berücksichtigung der Besonderheiten des Ziel-DBMS
  - ➔ Minimalisierung der Zugriffe bzw. der Zugriffszeiten

3.Das ENTITY-RELATIONSHIP-Modell

3.1Allgemein

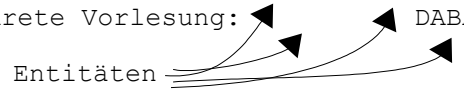
- 1976 von Peter Chen vorgeschlagen
- hat hohe Bedeutung erreicht, vor allem beim Datenbankentwurf
- ermöglicht eine grafische Darstellung
  - ➔ einfach
  - ➔ leicht verständlich

3.2 Elemente

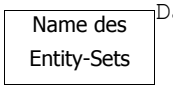
Entity (Entität)

- *wohl unterscheidbare Dinge der realen Welt*

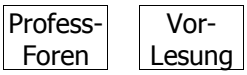
- Bsp: konkreter Professor: RÖSCH LANG
  konkrete Vorlesung: DABA DATÜ



zusammenhängende Entitäten werden zu einem Entity-Set (Entitäten-Menge) zusammengefasst. Grafische Darstellung:



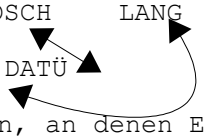
Beispiele:



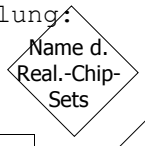
Relationship (Beziehung)

- *Häufig besteht zwischen Entitäten eine Beziehung*

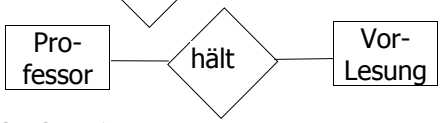
- Bsp: RÖSCH LANG
  DATÜ DABA



Beziehungen, an denen Entitäten vom gleichen Typ beteiligt sind, werden zu einem Relationship-Set (Beziehungsmengen) zusammengefasst.
Grafische Darstellung:



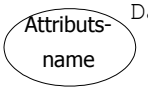
Beispiel



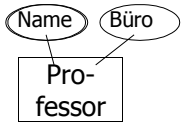
Attribute (Eigenschaften)

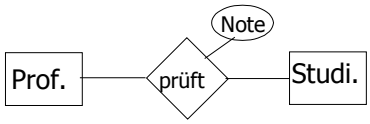
- *sowohl Entitäten, als auch Beziehungen können Attribute aufweisen*

Grafische Darstellung:



Beispiele

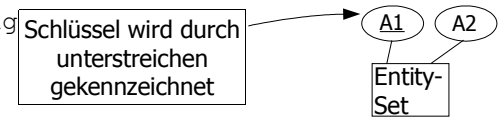




Schlüssel

- *minimale Menge von Attributen, deren Werte eine Entität eindeutig identifizieren*

Grafische Darstellung

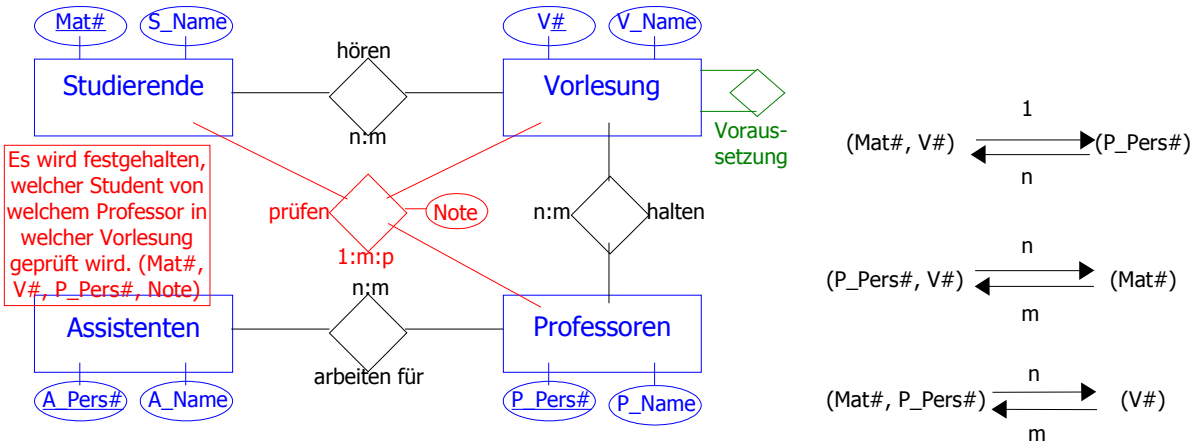


- Es kann mehrere Schlüsselkanditaten geben

Beispiel

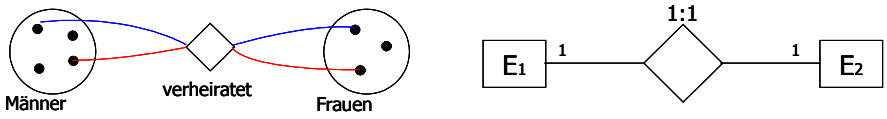
Miniwelt „FH“

- Es gibt viele Studierende
- Es gibt viele Professoren
- Es gibt viele Assistenten
- Es gibt viele Vorlesungen
- Professoren halten Vorlesungen
- Studierende hören Vorlesungen
- Assistenten arbeiten für die Professoren
- Studierende werden von einem Professor über den Stoff einer Vorlesung geprüft. Sie erhalten eine Note
- Es gibt Vorlesungen, die den Stoff einer anderen Vorlesung voraussetzen

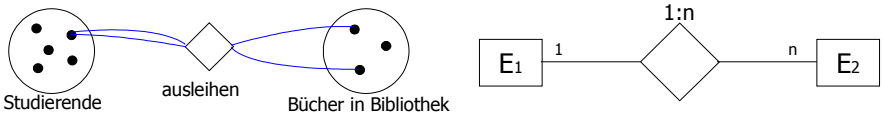


3.3 Kompatibilität von Beziehungen

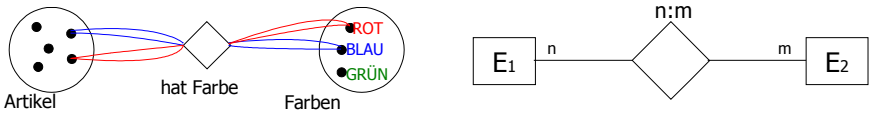
1:1 Beziehung: Jeder Entität e_1 aus der Entitätsmenge E_1 ist höchstens eine Entität e_2 aus E_2 zugeordnet (und umgekehrt).



1:n Beziehung: Jeder Entität e_1 aus E_1 können beliebig viele (mehrere oder auch keine) Entitäten aus E_2 zugeordnet sein. Einer Entität aus E_2 ist höchstens eine Entität aus E_1 zugeordnet.

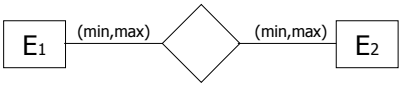


n:m Beziehung: Jeder Entität aus E_1 können beliebig viele Entitäten aus E_2 zugeordnet sein (und umgekehrt).



3.4 Minimum-Maximum Notation

Präzisere Angaben für die Komplexität von Beziehungen können über die (min-max)-Notation
gemacht werden. Es werden dabei genauere Ober- und Untergrenzen festgelegt.

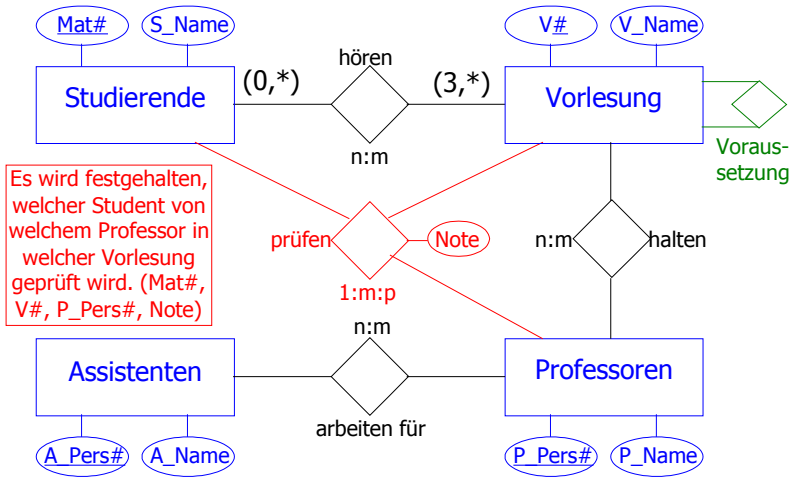


- *Sonderfälle*

1. Gibt es Entitäten, die mit keiner anderen in Beziehung stehen, schreibt man *min=0*
2. Gibt es Entitäten, die mit beliebig vielen in Beziehung stehen, schreibt man *max=**

Präzisierung für das FH-Beispiel

1. Vorlesung werden von mind. 3 Studierenden gehört (sonst findet sie nicht statt).
2. Assistenten sind genau einem Professor zugeordnet
3. Eine Vorlesung wird von genau einem Professor gehalten
4. Professoren können vom Vorlesungsbetrieb freigestellt werden



Beispiel Sportgeschäft

Ein Sportgeschäft bietet verschiedene Artikel an, wie „Jacke Rain" vom Typ Regenjacke.
Die Artikel weisen unterschiedliche Merkmale auf (wie „wasserdicht", „winddicht").
Sie existieren in unterschiedlichen Größen und Farben. Die Artikel werden von Produzenten
geliefert. Ein konkreter Artikel wird von genau einem Produzenten geliefert. Das Sportgeschäft
möchte über die
Datenbank wissen, es kann mehrere Artikel diesen Typs geben (z.B. Regenjacke); im Beispiel nur ein Prod./ Artikel
welche Artikel
auf Lager sind und welche nicht.

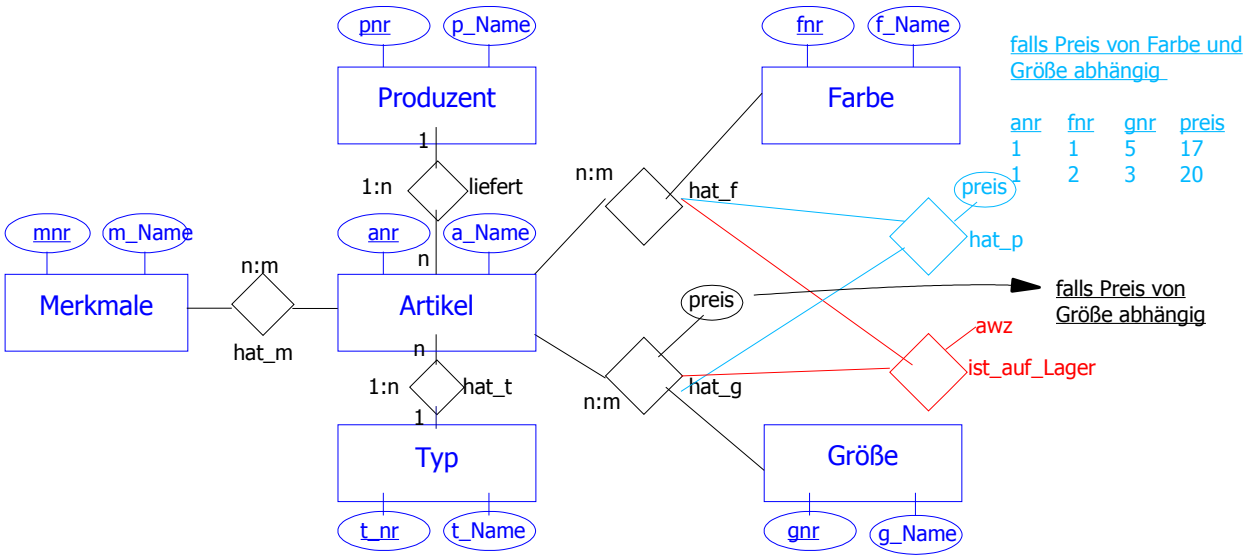
Artikel

Anr Name Typ Preis — verschiedene Größen können unterschiedliche Preise haben

auf Lager — z.B. grün=5 rot=1

Produzent — kann mehr als nur diesen einen Artikel liefern

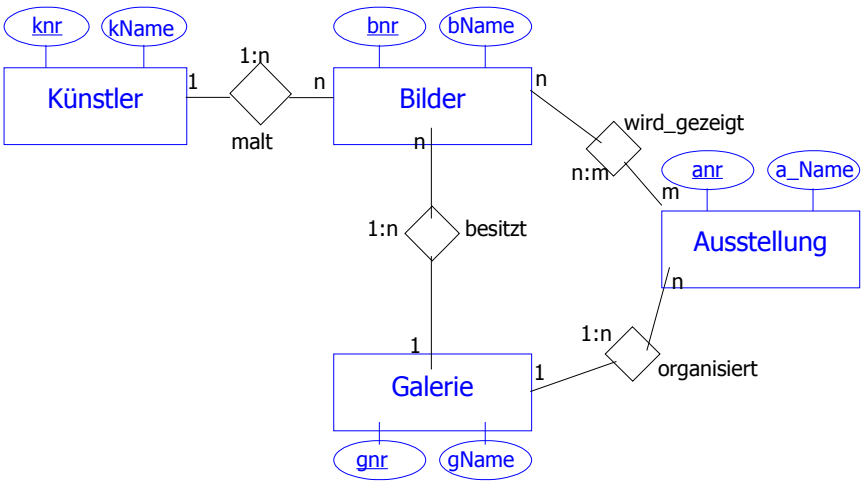
Merkmal — unterschiedliche Artikel können diese Merkmale (z.B. „winddicht") haben



Wie wird mit Artikeln umgegangen, die nicht auf Lager sind?
Wenn nur wenige Artikel in der Regel auf Lager sind, ist es sinnvoll, auch nur diese in `ist_auf_Lager` zu erfassen.

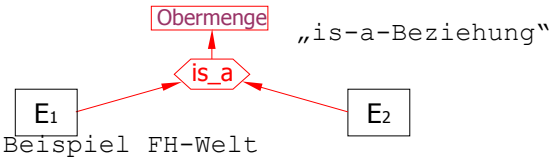
Beispiel Bildende Kunst

Es sollen Künstler verwaltet werden, die im Laufe ihres Lebens Bilder gemalt haben.
Diese befinden sich in Galerien. Es gilt: 1 Bild wurde von genau einem Maler gemalt
und befindet sich genau im Besitz einer Galerie. In Ausstellungen werden Bilder gezeigt.
Eine Ausstellung wird genau von einer Galerie durchgeführt.

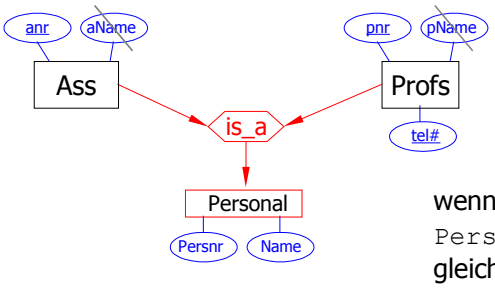


Generalisierung

- wird eingesetzt, um eine bessere Strukturierung der Entitätsmenge zu erreichen
- Gleiche Eigenschaften von verschiedenen Entitätsmengen werden in einer Obermenge zusammengefasst.
- verschiedene Eigenschaften verbleiben in der jeweiligen Entitätsmenge.
- Grafische Darstellung:



Beispiel FH-Welt

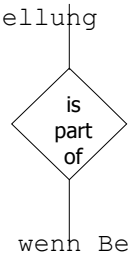


wenn nur Profs Telefon haben, dann ist tel# in Entitätsmenge Profs

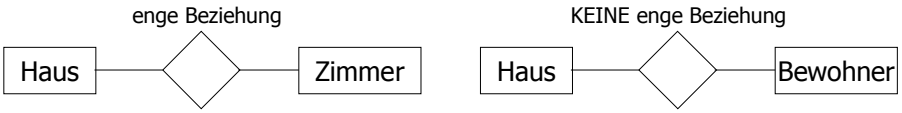
wenn anr und pnr in Obermenge als Persnr geführt sind, können sie nicht gleich sein.

Aggregation

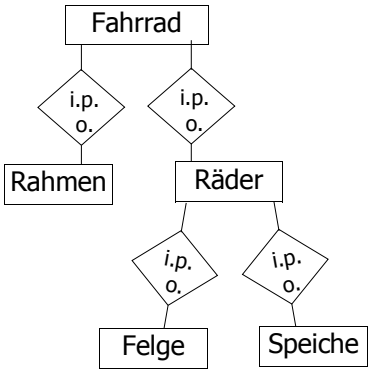
- wird eingesetzt, um unterschiedliche Entitätsmengen, die in ihrer Gesamtheit eine Obermenge bilden, einander zuzuordnen.
- Die Aggregation ist eine „is-part-of" Verknüpfung
- Grafische Darstellung



- wird verwendet, wenn Beziehung sehr eng ist (kann nicht ohne andere Entität existieren).



Beispiel



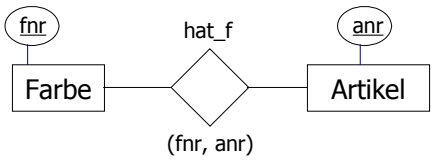
4. Logischer Entwurf

4.1Allgemein

Beim logischen Entwurf wird das ER-Diagramm in das Datenmodell des zu verwendendem DBMS umgesetzt. In der Praxis ist das relationale Datenmodell das gängigste. Für die Umsetzung gibt es Transformationsregeln, die konsequent angewendet werden müssen.

4.2Das relationelle Datenmodell

Alle Informationen werden über Relationen festgehalten. Speziell die Beziehungen werden über
inhaltliche Angaben über eine Relation festgehalten, z.B. steht **Farbe** mit **Artikel** in Beziehung.
Die Beziehung wird inhaltlich über **fnr** und **anr** festgehalten.



Zur Wiedergewinnung von Informationen gibt es 3 Operationen

- *Selektion (Auswahl bestimmter Zeilen)*
- *Projektion (Auswahl von Spalten)*
- *Join (Verbund) (Zeige Beziehung)*

4.3Tabellen und Relationen

- Was ist eine Relation?
  formal: Kartesisches Produkt von zwei Mengen M_1, M_2

$$M_1 \times M_2 = \{(m_1, m_2) | m_1 \in M_1, m_2 \in M_2\}$$

Beispiel

M_1 = {rot, blau, grün}
M_2 = {(JackeBlitz)(HoseTaifun)}

$M_1 \times M_2$ {(r,JB), (r, HT), (bl,JB), (bl,HT), (gn,JB), (gn, HT)}

Eine Relation R ist eine Teilmenge von $M_1 \times M_2$

Beispiel R={(r,JB),(gn,HT),(bl,HT)}

- Was ist eine Tabelle?

anr	aname	fnr
1	Jacke Blitz	rot
2	Hose Taifun	grün, blau

nicht atomar: bei einer Tabelle sind nicht atomare Spalten möglich, bei einer Relation nicht

Eine Tabelle entspricht einer Relation, wenn die Spalten der Tabelle nur atomare Werte aufweist.

4.4Transformationen nicht atomarer Attribute

Fall 1: **zusammengesetzte Attribute**

Beispiel: Ein zusammengesetztes Attribut wird durch seine Komponenten ersetzt

Name	Adresse
Meier	Bingen, Rochusallee 15

Transformation

Name	Wohnort	Straße
Meier	Bingen	Rochusallee 15

Fall 2: **mehrwertige Attribute**

Beispiel: artikel(anr, aname, farbe) farbe ist mehrwertig bzgl. anr

Die Relation ist in zwei Relationen aufzuteilen. Die eine Relation enthält das mehrwertige
Attribut und das Schlüsselattribut, die andere Relation enthält den Rest incl. Schlüsselattribut

artikel(anr, aname, farbe, preis)

artikel(anr, aname, preis)
hat_f(anr , farbe)

artikel(...)={(1,JackeB,blau,50),(2,HoseT,rot,200)}
grün

artikel={(1,JackeB,50),(2,HoseT,200)}
hat_f={(1,blau),(1,grün),(2,rot)}

Die mehrwertigen Attribute können nicht auftreten, wenn der konzeptionelle Entwurf korrekt gemacht wurde.

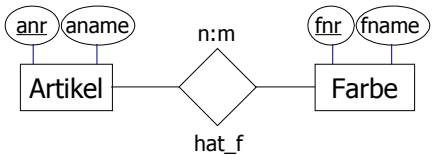
4.5Transformationsregeln

- *Regel 1: Entitäsmengen*

  Jede Entitäsmenge wird durch eine eigenständige Relation (Tabelle) mit einem eindeutigen Primärschlüssel umgesetzt.

- *Regel 2: Beziehungsmengen*

  Jede Beziehungsmenge kann als eigenständige Relation (Tabelle) definiert werden.
  Primärschlüssel der Beziehungsmenge ist der aus den Fremdschlüssel zusammengesetzte
  Schlüssel. Ein Fremdschlüssel wird gebildet durch den Primärschlüssel einer anderen
  Relation (Tabelle).



Relationen aus TrR 1 artikel(anr,aname)
farbe(fnr,fname)

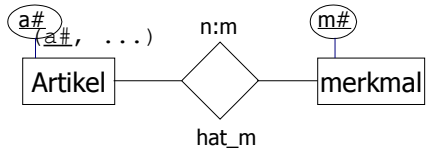
aus TrR 2 hat_f(anr,fnr)

anr ist Primärschlüssel in artikel
fnr ist Primärschlüssel in farbe

anr in der Relation hat_f ist ein Fremdschlüssel zur Relation artikel

- *Regel 3: n:m Beziehung*

  Jede n:m Beziehung muss als eigenständige Tabelle definiert werden. Der Primärschlüssel
  ist ein zusammengesetzter Schlüssel, wobei die Primärschlüssel der beteiligten Entitäsmengen
  als Fremdschlüssel verwendet werden.



nach Regel 1: artikel

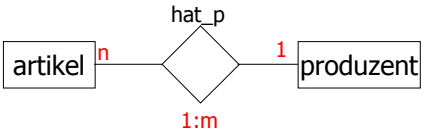
merkmal(m#, ...)

nach Regel 3: hat_m(a#,m#) eindeutige Werte (1,3)(1,3)(1,4)

Fremdschlüssel: Wert von a# in hat_f muss vorher in artikel definiert sein

- *Regel 4: 1:n Beziehung*

  Eine 1:n Beziehung kann als eigenständige Tabelle definiert werden. Falls 1:n Beziehungen
  ohne eigene Tabelle festgehalten werden soll, muss der Primärschlüssel derjenigen Tabelle
  in die andere Tabelle aufgenommen werden, für den die Mehrfachbeziehung gilt.

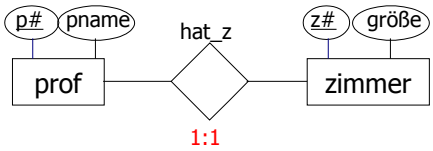


nach Regel 1: artikel(a#,aname,...)
produzent(p#,pname,...)

nach Regel 4: artikel(a#,aname,...,p#)
produzent(p#,pname,...) Fremdschlüssel

- *Regel 5: 1:1 Beziehung*

  Eine 1:1 Beziehung kann ohne eigenständige Tabelle festgehalten werden. Dazu wird der
  Schlüssel der einen Tabelle als Fremdschlüssel in die andere Tabelle aufgenommen.



nach Regel 1: prof(p#,aname,...)
zimmer(z#,größe,...)

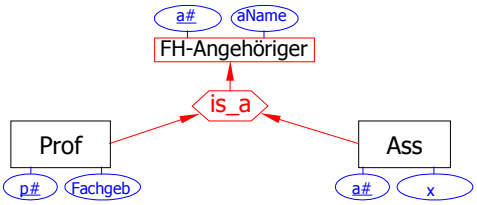
nach Regel 5: prof(p#,pname,...,z#) Möglichkeit 1
zimmer(z#,größe,...) Fremdschlüssel

prof(p#,pname,...) Möglichkeit 2
zimmer(z#,größe,...,p#) Fremdschlüssel

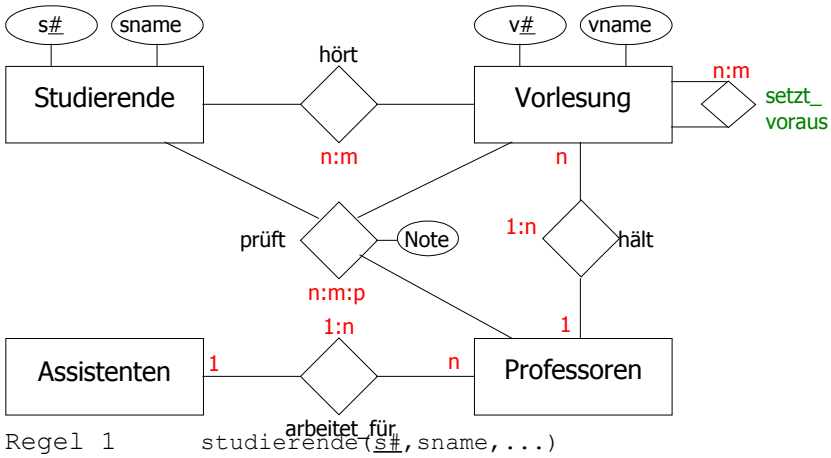
- *Regel 6: Generalisierung*

  Jede Entitäsmenge einer Generalisierungshierarchie wird eine eigenständige Tabelle. Der
  Primärschlüssel der übergeordneten Tabelle ist auch Primärschlüssel der untergeordneten Tabelle.

nach Regel 6: FH-Angehöriger(a#,aname,...)
prof(p#,fachgebiet,...)
ass(ass#,x,...) Fremdschlüssel

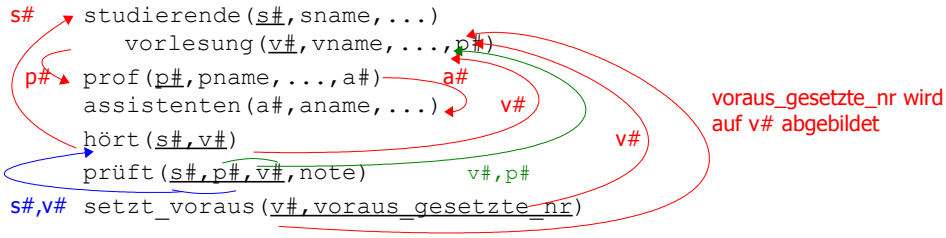


Beispiel FH-Miniwelt



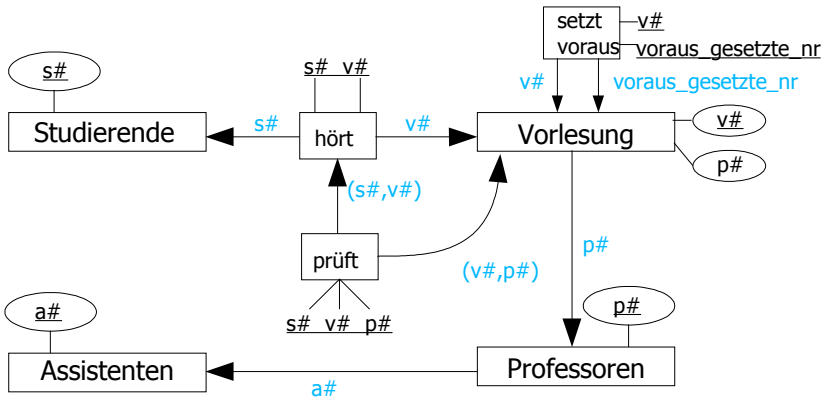
- Regel 1 studierende(s#,sname,...)
  vorlesung — wird ersetzt durch Regel 4 — (v#,vname,...)
  prof(p#,pname,...)
  assistenten (a#,aname,...)
- Regel 3 hört(s#,v#)
  prüft(s#,p#,v#,note)
  setzt_voraus(v#,vorausgesetzte_nr)
- Regel 4 vorlesung(v#,vname,...,p#)
  prof(p#,pname,...,a#)

Zusammenfassung



In prüft wird z.B. (1,2,3,1) eingetragen. DBMS überprüft, ob
s#=1 in studierende definiert ist
v#=2 in vorlesung definiert ist
p#=3 in prof definiert ist
Der Datensatz wird in prüft eingetragen, obwohl studierende die konkrete Vorlesung
evtl. noch gar nicht gehört hat. Daher muss in prüft ein zusammengesetzter Fremdschlüssel
definiert werden.

Diagramm



4.6Normalisierung

Ziel der Normalisierung ist die Optimierung des logischen Entwurfs (Optimierung der Relationen).
Prinzipien der Normalisierung können beim ER-Diagramm berücksichtigt werden.

- *Grundsätzliche Vorgehensweise:*
  - Feststellen der Datenabhängigkeiten
  - Falls welche auftreten, wird die betroffene Relation in geeignete kleinere Relationen zerlegt

Abhängigkeiten zwischen Attributen

Relation $R(A_1, A_2, \ldots, A_n)$

eine beliebige Kombination daraus:
Attribut Kombination $Y=(A_{i1}, A_{i2}, \ldots, A_{ik})$

Das Attribut A_v ist von Y funktional abhängig, wenn für jeden Wert von Y auf A_v geschlossen werden kann $Y \rightarrow A_v$ v=1...n

A_v heißt voll funktional abhängig von Y, wenn
$Y \rightarrow A_v$ d.h. funktional abhängig
Für jede $T \subset Y$: $T \rightarrow A_v$

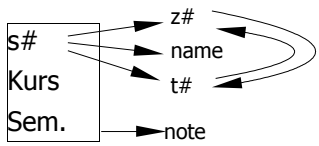
Av heißt transitiv abhängig von Y, wenn
$Y \rightarrow A_\mu$ es existiert ein μ
$A_\mu \rightarrow A_v$ $Y \rightarrow A_\mu \rightarrow A_v$
$A_\mu \not\rightarrow Y$

Beispiel
R(s#, name, z#, t#, kurs, sem, Note)

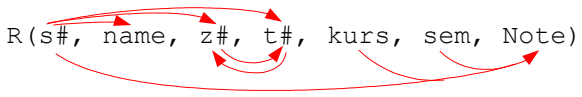
Zimmer Nr im Stud. Wohnheim	Tel Nr im Zimmer	Sem.	Sem., i.d. der Kurs geprüft wurde

Ermittlung der Abhängigkeiten

s#➔name
s#➔z#
s#➔t#
z#➔t#
t#➔z#



(s#, kurs, sem) ➔ note



Als **Relationenschlüssel** S einer Relation R wird das Attribut bzw. die Attributkombination
bezeichnet, für die gilt:

$$s \rightarrow (A_1, A_2, \ldots, A_n)$$

Für jede Teilmenge T von s $T \rightarrow (A_1, A_2, A_n)$ s ist minimal

Verfügt eine Relation über mehrere Relationenschlüssel, so heißen diese **Kanditatenschlüssel.**
Wählt man einen für das weitere aus, so heißt dieser **Primärschlüssel.**
Beispiel Primärschlüssel von R ist (s#, name, sem)

Wie findet man einen Relationenschlüssel? Reduktionsverfahren

Start: $K=(A_1, \ldots, A_n)$
Reduktionsschritt: 1.) Entferne ein A_i $(i=1...n)$
⇒ $K^*=K\backslash\{A_i\}$
2.) Überpüfe, ob A_i aus dem Rest (d.h. K^*) herleitbar ist.
3.) Falls ja:
- $K=K^*$
- wiederhole Reduktionsschritt

Falls nein:
- A_i muss in K bleiben
- wiederhole Reduktionsschritt mit anderem A_i
- Falls dies für alle A_i aus K scheitert, stellt K den gesamten Schlüssel dar

Beispiel

Start: K=(s#,name, z#, t#, kurs, sem, note)
1.Red.Schritt: s# entfernen
K*=(name, z#, t#, kurs, sem, note)
überprüfe K*→s# (Müller,1,2,DaBa,SS02,3) ⇒nein !
K=K* {s#}
2.Red.Schritt: name entfernen
K*=(s#, z#, t#, kurs, sem, note)
überprüfe K*→s# s#→name ⇒ja !
K=K*=(s#, z#, t#, kurs, sem, note)
3.Red.Schritt: z# entfernen
überprüfe K*=(s#, t#, kurs, sem, note) →z# ⇒ja !
K=K*
4.Red.Schritt: t# entfernen, überprüfe ⇒ja !
K=(s#, kurs, sem, note)
5.Red.Schritt: Kurs ⇒nein !
6.Red.Schritt: sem ⇒nein !
7.Red.Schritt: note ⇒ja !

⇒K=(s#, kurs, sem)

Hausaufgabe
R(A, B, C, D, E, F)
(A,B)→(D,F)
(A,D)→(F)
(D)→(B)
(F)→(C)

Lösung
(A, D, E) (A, B, E)

Man unterscheidet:

- *Primärattribute*

  sind alle Attribute, die mindestens in einem Kanditatenschlüssel enthalten sind

- *Sekundärattribute*

  alle anderen Attribute

Eine Relation R befindet sich in der ersten Normalform (1NF), wenn alle Attribute atomar sind. Eine Relation R befindet sich in der zweiten Normalform (2NF), wenn jedes Sekundärattribut von jedem beliebigen Schlüssel voll funktional abhängig ist.

Beispiel

R(s#, name, z#, t#, kurs, sem, note) Primärattribute

(s#, kurs, sem,)→name Diese Relation ist nicht in 2NF wegen s#→name

Normalisierung: existiert eine Attributkombination Y, die vom Schlüssel S nur partiell abhängig ist, so wird Y mit dem partiellen Schlüsselteil in eine eigene Relation projeziert.

Beispiel

R(s#, name, z#, t#)

kann Anomalien verursachen

R(s#, kurs, sem, note) R_1 & R_2 sind 2NF

Eine Relation R befindet sich in der dritten Normalform (3NF), falls alle Sekundärattribute von jedem Schlüssel direkt und transitiv abhängig sind.

Normalisier ung:

R(S, X, Y, Z)
S → X → Y

$R_1(\underline{X}, Y)$
$R_2(\underline{S}, X, Z)$

$R(A_1, A_2, \ldots, A_n)$
$S=(A_{i1}, A_{i2}, \ldots, A_{ik})$ $Z=(A_{j1}, \ldots, A_{jp})$ $j_1, j_p \notin \{i_1, \ldots, i_x\}$
$S \rightarrow A_j \rightarrow A_m$ $j,m \notin \{i_n, \ldots, i_k\}$

$R_1(A_j, A_m)$
$R_2(A_{i1}, \ldots, A_{jk}, A_j, A_{j1}, \ldots, A_{jp})$

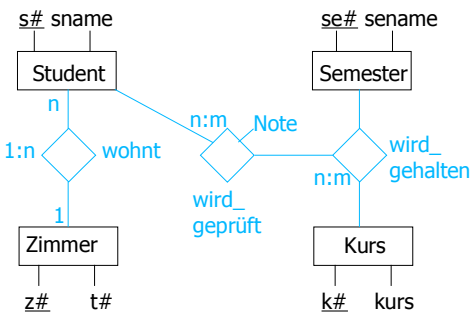
S X Z

R_{11}(s#, sname, z#)

R_{12}(z#, t#)

R_2(s#, kurs, sem, note)

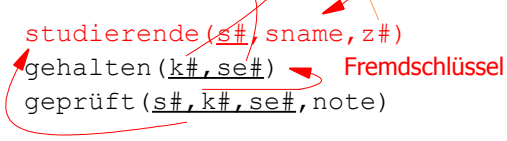
Entstehen die gleichen Relationen, wenn mit ER-Diagramm gearbeitet wird?



Transformation

1. studierende(s#, sname)
   zimmer(z#, t#)
   kurs(k#,kurs)
   sem(se#,sem)

2. studierende(s#,sname,z#)
   gehalten(k#,se#)
   geprüft(s#,k#,se#,note)

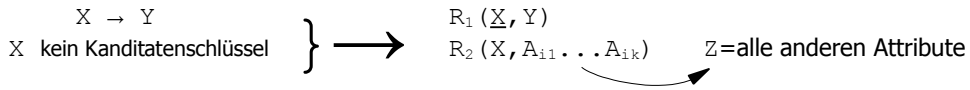


Zusammenfassung:

studierende(s#,sname,z#) ≙ R_{11}
zimmer(z#, t#) ≙ R_{12}
geprüft(s#,k#,se#,note) ≙ R_2

Boyce-Codd-Normalform (BCNF)

- Untersuchung der funktionalen Abhängigkeit zwischen Primärattributen
- Definition der Determinante: A heißt Determinante von D, wenn gilt
  1. A B funktional abhängig
  2. ∀ T⊂A T↛B
- Definition der Boyce-Codd´schen Normalform
  Eine Relation ist in der BCNF, wenn jede Determinante der Relation ein Kanditatenschlüssel ist.
- Normalisierung R $(A_1, \ldots, A_n)$



Beispiel

R(s#, name, z#, t#, kurs, sem, Note)

Determinante bestimmen:

s#	weil s# →	sname, z#, t#
s#,kurs,sem	weil	(s#,kurs,sem)→note
z#	weil	z#→t#
t#	weil	t#→z#

Kanditatenschlüssel bestimmen
(s#,kurs,sem)

R ist nicht in BCNF, weil es 3 Determinanten gibt, die keinen Kanditatenschlüssel darstellen.

Normalisieren:

1. z#→t#
   z# kein Kanditatenschlüssel
   } R_1(z#,t#)
   } R_2(s#,sname,z#,kurs,sem,note)

R_1 in BCNF, R_2 nicht

2. s# → sname, z#
   s# kein Kanditatenschlüssel
   } R_{21}(s#,sname,z#)
   } R_{22}(s#,kurs,sem,note)

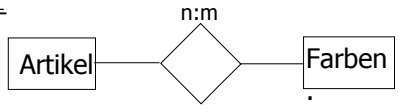
R_{21} in BCNF, R_{22} in BCNF

Vierte Normalform (4NF)

1) Definition: mehrfach abhängig

- In einer Relation R(A_1,...,A_n) heißt B mehrfach abhängig von A, wenn zu jedem A
  mehrere Werte von B exitieren.
  A-->>B

Beispiel



hat_f(a#,f#) a#-->>f#

a#	f#
1	2
1	3

In einer Relation R(...) liegt eine nicht triviale mehrfache Abhängigkeit vor, wenn folgendes gilt

A-->>B
A-->>C

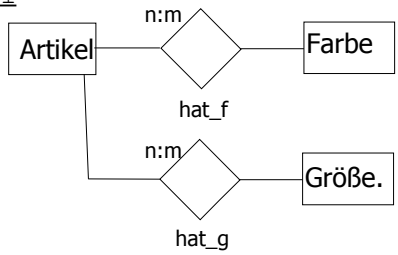
Definition 4NF: Eine Relation befindet sich in 4NF, wenn sie in der BCNF und keine
nicht trivialen Abhängigkeiten auftreten

Normalisierung

R(A,B,C,Y)
A-->>B
A-->>C

→ R_1(A,B), R_2(A,C), R_3(A,Y)

Beispiel



Anderer Ansatz: farbe & größe zusammen in eine Entitäsmenge

anr	fnr	gnr
1	1	1
1	1	2
1	1	3
1	2	2
1	2	4

anr-->>fnr

anr-->>gnr

⇒ Normalisierung
R_1(anr,fnr)
R_2(anr,gnr)
R_3(anr,...)

Umsetzung der Relationen in die DDL

DDL (*data description language*) ist eine Teilmenge der SQL-Sprache. Zu der DDL gehören
z.B. create table, mit dem Tabellen eingerichtet werden können oder create user, mit dem Benutzer eingerichtet werden können. Mit create table hat man die Möglichkeit :

1) Spalten und deren Datentypen zu definieren
2) Semantische Integrität zu definieren
3) Entitäs-Integrität zu definieren } Constraint-Bedingungen
(Integritäts-Bedingungen)
4) Referentielle Integrität zu definieren

Es sind folgende Datentypen möglich :

- char (size) size gibt an, wie groß das Zeichenarray sein soll
- varchar (size) Zeichenarray der max. Länge size, ansonsten ist die Länge variabel
- long Zeichenstring von variabler Länge bis zu 6 GByte
- number (p,s) Zahl mit p Ziffern, davon s Ziffern hinter dem Komma
- integer Intergerzahl
- raw (size) Binärdaten bis zu size Bytes
- date Datum

Beispiel für das einfache Anlegen einer Tabelle :

```
                  Tabellen-Name      Spaltenname  Gewählter Datentyp
create table produzent (pnr    integer,
                        pname  char(20))
```

Bei diesem Aufruf fehlen sämtliche Integritätsangaben

Semantische Integrität:

- not null in der Spalte sind keine Nullstellen erlaubt (d.h. es muss ein Wert in der Spalte eingetragen werden.)
- null Nullwerte erlaubt
- default Setzen eines default-Wertes
- check Überprüfen eines Wertes auf Bedingungen
- unique Nullwerte erlaubt. sonst nur eindeutige Werte

Beispiel

lager (anr,fnr,gnr,anz)

wenn farblose Artikel möglich sind, dann (anr,fnr,gnr) als unique vereinbaren.

(1, ,2,7) (3, ,2,7)

(2,3,2,8) (1, ,5,8)

Entitätsintegrität: Primärschlüssel
Referentielle Integrität: Fremdschlüssel

Integritätsbedingungen können definiert werden als

- *Spaltenbezogene Bedingungen (Column constraint)*

  Bedingungen, die sich eindeutig auf eine Spalte beziehen produzent (pnr,pname)

- *Tabellenbezogene Bedingungen (table constraint)*

  Bedingungen, die sich auf mehrere Spalten beziehen hat_f(pnr,fnr)

Beispiel, bei dem für die Constraints kein Name vergeben wird :

```
create table produzent (pnr   integer primary key,
                        pname  char(20) not null)

create table produzent (pnr integer constraint PK_produzent primary key,
                        pname char(20) constraint NN_produzent not null)
```

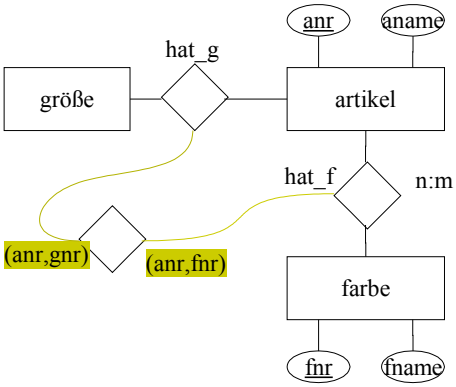
```
artikel(anr,aname,pnr)

create table artikel(anr integer constraint PK_artikel primary key,
                     aname integer constraint FK_artikel_produzent
                     references produzent, on delete cascade)
```

Automatisches Löschen in der untergeordneten Tabelle (artikel()), wenn in der übergeordneten Tabelle (produzent) ein Satz gelöscht wird

produzent	
pnr	pname
~~1~~	~~nike~~
2	addidas

artikel	
anr,aname,pnr	
~~1~~	~~1~~
2	2




```
artikel(anr, aname)
hat_f(anr,fnr)
farbe(fnr,fname)
```

⇓

```
create table artikel(anr integer primary key,
                     aname char(20) not null)

create table farbe(fnr integer primary key,
                   fname char(20) not null)

create table hat_f(anr integer references
                   artikel, fnr integer
                   references farbe,
                   primary key(arn,fnr))
```

Andere Möglichkeit:

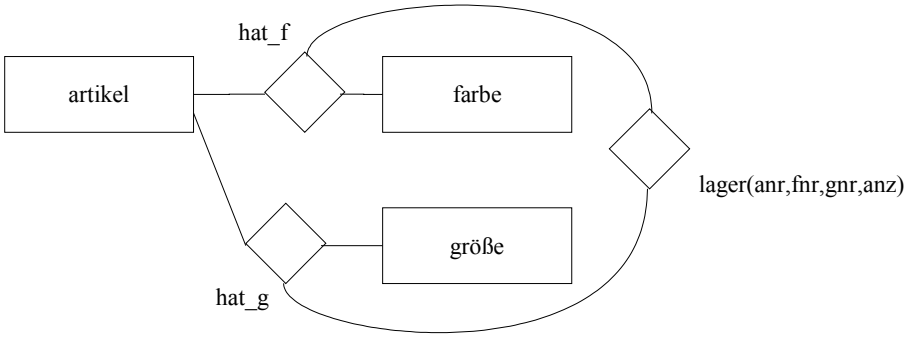
(**Fremdschlüssel**angabe)

```
create table artikel(anr integer,
                     fnr integer,
                     primary key(anr,fnr),
                     foreign key(anr) references artikel(anr),
                     foreign key(fnr) references farbe)
```

```
create table lager(anr integer,
                   fnr integer,
                   gnr integer,
                   anz integer check(anz>0),
                   primary key(anr,fnr,gnr),
                   foreign key(anr,fnr) references hat_f,
                   foreign key(anr,gnr) references hat_g)
```

anr	fnr	gnr	anz
1	1	1	10

Farblose Artikel existieren auf Lager:




```
create table lager(anr integer not null,
                   fnr integer,
                   gnr integer,
                   anz integer check(anz>0),
                   unique(anr,fnr,gnr),
                   foreign key(anr) references artikel,
                   foreign key(anr,fnr) references hat_f,
                   foreign key(anr,gnr) references hat_g)
```

lager(anr,fnr,gnr,anz)

anr	fnr	gnr	anz
1	1	1	10
1	NULL	1	5
FALSCH			
2	NULL	NULL	8

Es ist ein Artikel eingetragen, dessen Farbe mit NULL eingetragen ist. Dieser Artikel ist aber kein farbloser Artikel, sondern ein Artikel, den es in 2 Farben gibt (fnr=1 rot, fnr=2 blau)

keine Fehlermeldung (unique)