# Teil 8: Einführung in das Entity-Relationship-Modell

**Literatur:**


- Elmasri/Navathe:Fundamentals of Database Systems, 3. Auflage, 1999. Chapter 3, “Data Modeling Using the Entity-Relationship Model”
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3. Auflage, Ch. 2, “Entity-Relationship Model”.
- Ramakrishnan: Database Management Systems, Mc-Graw Hill, 1998, Ch. 14, “Conceptual Design and the ER-Model”
- Kemper/Eickler: Datenbanksysteme, Kapitel 2, Oldenbourg, 1997.
- Rauh/Stickel: Konzeptuelle Datenmodellierung, Teubner, 1997.
- Teorey: Database Modeling and Design, 3.Auflage, 1999.
- Barker: CASE*Method, Entity Relationship Modelling, Oracle/Addison-Wesley, 1990.
- Lipeck: Skript zur Vorlesung Datenbanksysteme, Univ. Hannover, 1996.






## Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- die drei Phasen des Datenbank-Entwurfs erklären,

  Wozu sind verschiedene Phasen nützlich?

- die Bedeutung des ER-Modells für den DB-Entwurf erläutern,
- grundlegende Elemente des ER-Modells aufzählen,
- ER-Diagramme (Schemata im ER-Modell) für eine gegebene (kleine) Anwendung entwickeln,
- gegebene ER-Diagramme vergleichen/bewerten.





## Inhalt







## Datenbank-Entwurf (1)

- Ziel: Entwicklung von Programmen, um gegebene Aufgaben der realen Welt zu bearbeiten.
- Diese Programme benötigen persistente Daten.
- Verwendung von Software Engineering Methoden (aber spezialisiert auf datenintensive Programme).
- DB-Entwurf ist der Prozess der Entwicklung eines DB-Schemas für eine gegebene Anwendung.

  Es ist eine Teilaufgabe des allgemeinen Software Engineering.

# Datenbank-Entwurf (2)

- Die Anforderungen an Programme und Daten sind verflochten, beide hängen voneinander ab:
  - ◇ Das Schema sollte die Daten enthalten, die die Programme benötigen.
  - ◇ Die Programme sind meist leicht zu spezifizieren, wenn die zu verwaltenden Daten festliegen.
- Daten sind eine unabhängige Ressource:
  - ◇ Oft werden später zusätzliche Programme, die auf den vorhandenen Daten beruhen, entwickelt.
  - ◇ Auch ad-hoc-Anfragen sind möglich.





# Datenbank-Entwurf (3)

- Während des DB-Entwurfs muss ein formales Modell von Teilen der realen Welt ("Miniwelt") erstellt werden.

  Fragen über die reale Welt sollten von der Datenbank beantwortet werden. Eine Liste solcher Fragen kann ein wichtiger Input für den DB-Entwurf sein.

- Die reale Welt ist ein Maß für die Korrektheit des Schemas: Datenbank-Zustände sollten möglichen Situationen der realen Welt entsprechen.





# Datenbank-Entwurf (4)

- Datenbank-Entwurf ist nicht leicht:
  - ◇ Der Entwickler muss den Anwendungsbereich kennenlernen.

    U.a. führt man dazu Gespräche (Interviews) mit Anwendundungsexperten (z.B. spätere Nutzer der Datenbank). Diese nennen eventuell für sie selbstverständliche Dinge nicht.

  - ◇ Ausnahmen: Die reale Welt ist sehr flexibel.
  - ◇ Größe: DB-Schemata können sehr groß sein.
- Wie jede schwierige Aufgabe wird der DB-Entwurf in mehreren Schritten durchgeführt.





# Datenbank-Entwurf (5)

- Es gibt drei Phasen des DB-Entwurfs:
  - ◇ <u>Konzeptioneller DB-Entwurf</u> erstellt ein Modell der Miniwelt in einem konzeptionellen Datenmodell (z.B. dem Entity-Relationship Modell).

    Statt "konzeptionell" kann man auch "konzeptuell" sagen.

  - ◇ <u>Logischer DB-Entwurf</u> transformiert das Schema in das Datenmodell des DBMS.

    Dies ist heute fast immer das relationale Modell.

  - ◇ <u>Physischer DB-Entwurf</u> hat Verbesserung der Performance des endgültigen Systems zum Ziel.

    Indexe und Speicherparameter werden in dieser Phase gewählt.

# Datenbank-Entwurf (6)

Warum verschiedene Entwicklungsphasen?

- Probleme können getrennt und nacheinander abgearbeitet werden.
- Z.B. muss man während der konzeptionellen Entwicklung nicht über die Performance oder über Einschränkungen verschiedener DBMS nachdenken.

  Im Mittelpunkt: Entwicklung eines korrekten Modells der realen Welt.
- DBMS-Features beeinflussen den konzeptionellen Entwurf nicht (und nur teilweise den logischen).

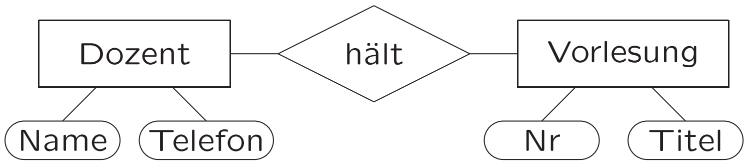
  Somit wird der konzeptionelle Entwurf nicht ungültig, wenn später ein anderes DBMS verwendet wird.





# Beispiel (1)

ER-Schema in graphischer Notation:



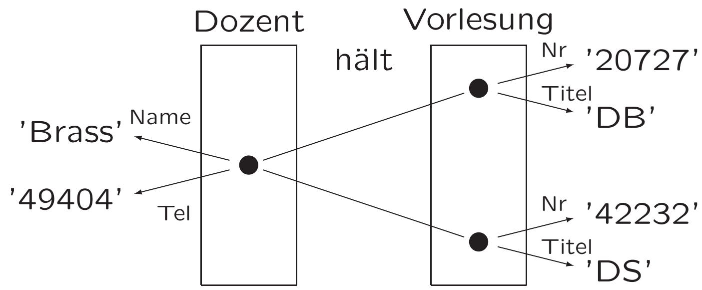

- Diese Miniwelt enthält Dozenten und Vorlesungen.
- Dozenten halten Vorlesungen.
- Dozenten haben Name und Telefonnummer.
- Vorlesungen haben Nr (z.B. "20727") und Titel.





# Beispiel (2)

Möglicher Zustand:








# Das ER-Modell (1)

- Das Entity-Relationship-Modell wird "semantisches Datenmodell" genannt, weil es die reale Welt genauer widerspiegelt als z.B. das relationale Modell.
  - ◇ Im ER-Modell werden Personen modelliert, im relationalen Modell nur ihre Namen/Nummern.
  - ◇ Im ER-Modell wird zwischen Entities und Relationships (Beziehungen) unterschieden. Im relationalen Modell wird beides in Tabellen dargestellt.

    Diese Aussagekraft wird nicht benötigt, um dem Informationsbedarf der Anwendungen zu genügen. Aber es verdeutlicht den Zusammenhang zwischen dem Schema und der realen Welt.

# Das ER-Modell (2)

- Vorgeschlagen von Peter Pin-Shan Chen (1976).
- Beinhaltet eine nützliche graphische Notation, die eine bessere Übersicht ermöglicht; um die Struktur der Daten zu "sehen".

  Dies unterstützt auch die Kommunikation mit zukünftigen Nutzern.
- Es gibt kein kommerzielles ER-DBMS.

  Eine Transformation des Schemas in ein anderes Datenmodell ist unvermeidbar. Objektorientierte DBMS sind jedoch ziemlich ähnlich.
- Viele Variationen und Erweiterungen des ER-Modells wurden vorgeschlagen.





# Das ER-Modell (3)

- Es gibt spezielle graphische Editoren und andere Entwicklungs-Tools.

  Z.B. ist Oracle Designer ein CASE-Tool für DB-Anwendungen und ein Bestandteil ist der ER Diagrammer. (CASE: Computer-Aided Software Engineering.) Alternativen: Sybase PowerDesigner, CA ERwin, ...
- Das ER-Modell ist ein Standard-Tool für den konzeptionellen DB-Entwurf. In letzter Zeit werden jedoch auch objektorientierte Methoden verwendet.

  Es könnte sein, dass UML (Unified Modelling Language) "die Zukunft" ist. Aber auch UML basiert auf dem ER-Modell. Kenntnis des ER-Modells ist eine wichtige Grundlage für jeden DB-Entwerfer.





# Inhalt







# Begriffe des ER-Modells (1)

Entities:

- Objekte der Miniwelt, über die Informationen gespeichert werden sollen. Z.B. Personen, Bücher, ...

  Es ist egal, ob Entities eine physische Existenz haben (und angefasst werden können) oder nur eine konzeptionelle Existenz.
- Die zu modellierende Miniwelt kann immer nur eine endliche Anzahl von Entities haben.

  Z.B. "alle Zahlen" (unendlich viele) können keine Entities sein.
- Es muss möglich sein, Entities voneinander zu unterscheiden, d.h. sie müssen eine Identität haben.

  Ameisen in einem Haufen können also keine Entities sein.

# Begriffe des ER-Modells (2)

Datentyp-Elemente:

- Werte einer möglicherweise unendlichen Menge, die gespeichert und ausgegeben werden können.
- Z.B. Strings, Zahlen, Datumswerte, Bilder.
- Eine Person kann nicht gespeichert werden (Entity), aber ihr Name (Datentyp-Element).
- Die meisten derzeitigen DBMS unterstützen eine vordefinierte Menge an Datentypen.
- Möglich: Non-Standard-Datentypen im ER-Schema verwenden (erschwert späteren logischen Entwurf).





# Begriffe des ER-Modells (3)

Attribute:

- Eine Eigenschaft oder Charakteristik eines Entities.

  Auch Relationships können Attribute haben, siehe unten.
- Z.B. der Titel dieser Vorlesung ist "Datenbanken I"
- Der Wert eines Attributs ist ein Element eines Datentyps, wie String, Integer, Datum: Er hat eine druckbare Darstellung.





# Begriffe des ER-Modells (4)

Relationship:

- Beziehung zwischen Paaren von Entities.

  Solche Relationships werden auch "binäre Relationships" genannt, um sie von solchen zu unterscheiden, bei denen mehr als zwei Entities beteiligt sind. Manche Autoren (aber wenige Tools) erlauben beliebige Relationships (z.B. ternäre Relationships). Aus meiner Erfahrung führt das oft zu Fehlern (Studenten "verschmelzen" zwei binäre Relationships zu einem ternären, das verletzt Normalformen).
- Z.B. Ich (Person) halte "DB I" (Vorlesung).
- Das Wort "Relationship" wird auch als Abkürzung für "Relationship-Typ" verwendet (siehe unten).

  Es sollte vom Kontext her klar sein, was gemeint ist.





# Begriffe des ER-Modells (5)

Entity-Typ:

- Menge ähnlicher Entities (in Bezug auf die zu speichernden Informationen), d.h. Entities, die die gleichen Attribute haben.
- Z.B. alle Dozenten dieser Universität.

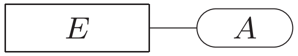
Relationship-Typ:

- Menge ähnlicher Relationships.
- Z.B. "X hält Vorlesung Y".

# ER-Diagramme (1)

- Entity-Typ $E$:


- Attribut $A$ des Entity-Typs $E$:



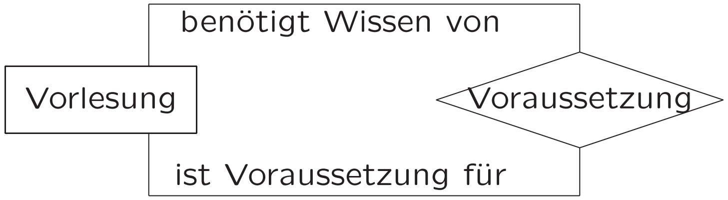

- Relationship $R$ zwischen Entity-Typen $E_1$ und $E_2$:






# ER-Diagramme (2)

- Relationships können auch zwischen Entities des selben Typs existieren ("rekursive Relationships"):




- In diesem Fall müssen an den Verbindungslinien "Rollennamen" zugefügt werden.

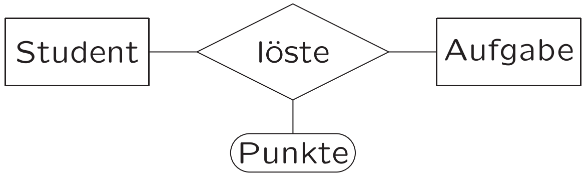
  Rollen können auf diese Art für jedes Relationship verwendet werden.





# ER-Diagramme (3)

- Relationships können auch Attribute haben:




- Das bedeutet, dass eine Punktzahl für jedes Paar Student X und Aufgabe Y gespeichert wird, so dass X eine Lösung für Y abgegeben hat.





# Graphische Syntax (1)

- Ein ER-Diagramm enthält
  - ◇ Rechtecke,
  - ◇ Rauten,
  - ◇ Ovale

  und Verbindungslinien.
- Rechtecke, Rauten und Ovale sind jeweils mit einer Zeichenkette markiert.

  Diese Zeichenkette wird in das Element geschrieben.

# Graphische Syntax (2)

- Verbindungslinien sind nur erlaubt zwischen
  - ◇ einem Rechteck und einer Raute,
  - ◇ einem Rechteck und einem Oval und
  - ◇ einer Raute und einem Oval.
- Außerdem müssen folgende Bedingungen gelten:
  - ◇ Eine Raute muss genau zwei Verbindungslinien zu Rechtecken haben. Jede Anzahl von Verbindungslinien zu Ovalen ist erlaubt.
  - ◇ Ein Oval muss genau eine Verbindungslinie haben.





# Graphische Syntax (3)

- Die Namen der Rechtecke (Entity-Typen) müssen im ganzen Diagramm eindeutig sein.

  D.h. es kann nicht zwei Rechtecke mit der selben Aufschrift geben.
- Die Namen der Ovale (Attribute) müssen nur für ein Rechteck oder eine Raute eindeutig sein.

  D.h. es kann nicht zwei Ovale mit dem selben Namen geben, die mit dem gleichen Rechteck (oder mit der gleichen Raute) verbunden sind.
- Rauten (Relationships) müssen durch Name und verbundene Rechtecke eindeutig identifiziert sein.

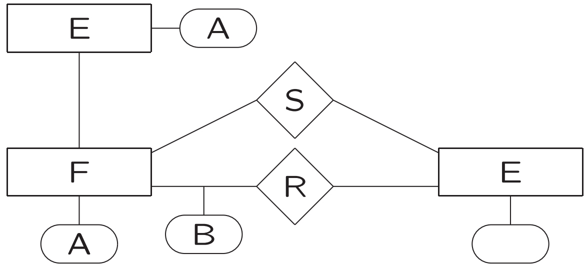
  D.h. es kann nicht zwei gleichnamige Rauten geben, die mit den gleichen Rechtecken verbunden sind.





# Übung

Welche Fehler enthält dieses Diagramm?








# Übung

Definieren Sie ein ER-Schema (Diagramm) für die folgende Anwendung:

- Informationen über Forscher im Gebiet Datenbanken sollen gespeichert werden.
- Für jeden Forscher wird Nachname, Vorname, E-Mail-Adresse und Homepage (URL) benötigt.
- Auch der derzeitige Arbeitgeber wird benötigt (Annahme: alle Forscher arbeiten an einer Uni).
- Für jede Universität sollen der Name, die URL und das Land gespeichert werden.

# ER-Schemata (1)

- Gewöhnlich werden nicht alle Schema-Informationen im ER-Diagramm dargestellt:
  - ◇ Attribute haben Datentypen, die in vollständigen ER-Diagrammen festgelegt werden müssen.
  - ◇ Manchmal wird ein ER-Diagramm deutlicher, wenn die Attribute nicht angezeigt werden.

    Es ist nicht ungewöhnlich, dass ein Entity-Typ 50 Attribute hat.
  - ◇ Kommentare/Erklärungen sind nützlich, sehen aber in ER-Diagrammen nicht gut aus.





# ER-Schemata (2)

- Es gibt also ein "wirkliches Schema" (z.B. in Textform oder in einer DB). ER-Diagramme sind nur Auszüge, die zur Illustration genutzt werden.

  Es gibt keinen Standard für eine Syntax, um ganze Schemata aufzuschreiben, dagegen gibt es für ER-Diagramme bekannte Notationen.
- Wichtig ist, die graphische Syntax zu lernen (und dass möglicherweise nicht alles dargestellt wird).
- Moderne Entwicklungs-Tools lösen das Problem: Z.B. ein Klick auf einen Entity-Typ im Diagramm zeigt weitere Informationen in einer Dialog-Box.





# ER-Schemata: Semantik (1)

Ein DB-Zustand $\mathcal{I}$ interpretiert die Symbole im Schema durch Definition einer

- endlichen Menge $\mathcal{I}[E]$ für jeden Entitity-Typ $E$,
- Abbildung $\mathcal{I}[A]\colon \mathcal{I}[E] \to val(D)$ für jedes Attribut $A$ eines Entity-Typs $E$, wobei $D$ der Datentyp von $A$ und $val(D)$ die Domain (Wertemenge) von $D$ ist,
- Relation $\mathcal{I}[R] \subseteq \mathcal{I}[E_1] \times \mathcal{I}[E_2]$ für jedes Relationship $R$ zwischen Entity-Typen $E_1$ and $E_2$,
- Abbildung $\mathcal{I}[A]\colon \mathcal{I}[R] \to val(D)$ für jedes Attribut $A$ eines Relationships $R$ ($D$ ist der Datentyp von $A$).





# ER-Schemata: Semantik (2)

- Dies ist ein Spezialfall der logischen Interpretation, die in Kapitel 3 vorgestellt wurde.

  Nur Relationship-Attribute sind schwierig mit Standard-Logik zu behandeln. Interpretation der Datentypen vom DB-Zustand getrennt.
- Die Haupteinschränkungen des ER-Modells sind:
  - ◇ Es können nur neue Sorten mit endlichen Domains eingeführgeführt werden (Entity-Typen).
  - ◇ Neue Funktionen nur von Entity-Typen (oder Relationships) zu Datentypen (Attribute).
  - ◇ Neue Prädikate müssen 2 Entity-Typen als Argumente haben (Relationships).

# ER-Schemata: Semantik (3)

Beispiel-Zustand:

- $\mathcal{I}[Dozent] = \{sb, ms\}$
- $\mathcal{I}[Name] = f_1$, mit
  $f_1(sb) =$ 'Brass', $f_1(ms) =$ 'Spring'.
- $\mathcal{I}[Tel] = f_2$, mit
  $f_2(sb) = 49404$, $f_2(ms) = 49429$.





# ER-Schemata: Semantik (4)

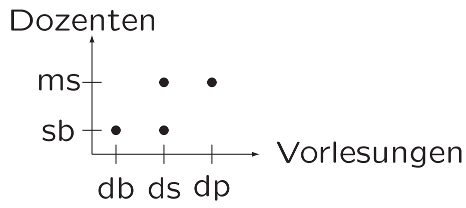
- $\mathcal{I}[Vorlesung] = \{db, ds, dp\}$
- $\mathcal{I}[Nr] = g_1$, mit
  $g_1(db) = 20727$, $g_1(ds) = 42232$, $g_1(dp) = 40492$.
- $\mathcal{I}[Titel] = g_2$, mit
  $g_2(db) =$ 'Datenbankverwaltung',
  $g_2(ds) =$ 'Datenstrukturen',
  $g_2(dp) =$ 'Document Processing'.





# ER-Schemata: Semantik (5)

- $\mathcal{I}[hält] = \{(sb, db),\ (sb, ds),\ (ms, ds),\ (ms, dp)\}$.
- $(X, Y)$-Paare mit Dozent $X$ und Vorlesung $Y$:








# ER-Schemata: Semantik (6)

Konsequenzen für Attribute:

- Es gibt Erweiterungen, aber im grundlegenden ER-Modell gilt folgendes:
  - ◇ Attributwerte sind immer definiert.

    Keine unbekannten Werte.
  - ◇ Attribute sind einwertig.

    Keine mengenwertigen Attribute.
  - ◇ Attribute sind atomar.

    Keine Record-Strukturen (oder Datentypen haben schon diese Struktur, z.B. Datumswerte). Das Basis-ER-Modell behandelt alle Attributwerte als atomar, d.h. es fügt keine Record-Strukturen außer den bereits durch die Datentypen gegebenen hinzu.

# ER-Schemata: Semantik (7)

Konsequenzen für Relationships:

- Es kann für einen Relationship-Typ nicht mehrere Verbindungen zwischen den gleichen zwei Entities geben.

  Ein Relationship wird durch eine Menge von Entity-Paaren interpretiert. Eine Menge enthält entweder ein Element oder sie enthält es nicht. Es kann nicht "zweimal" enthalten sein.

- Wenn das Relationship also ein Attribut hat, muss der Wert für jedes Entity-Paar eindeutig sein.

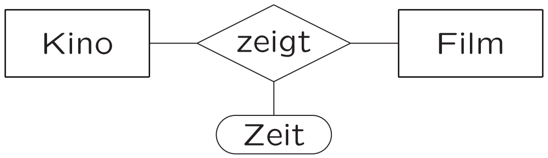




# ER-Schemata: Semantik (8)

Beispiel/Übung:

- Betrachten Sie dieses Schema:




- Stellen Sie sich vor, ein Kino zeigt den gleichen Film um 15 Uhr und um 18 Uhr.
- Kann man dies im gegebenen Schema speichern?

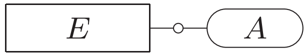
  ☐ Ja ☐ Nein





# Optionale Attribute

- Man kann auch Attribute zulassen, die nur teilweise definiert sind (optional, können NULL sein).
- Solche Attribute können mit einem kleinen Kreis auf der Linie zwischen Entity-Typ (oder Relationship) und Attribut markiert werden:




- Die Semantik dieses Attributs in einem Zustand $\mathcal{I}$ ist eine Teilfunktion von $\mathcal{I}(E)$ (Menge aller Entities) nach $val(D)$, wobei $D$ der Datentyp von $A$ ist.





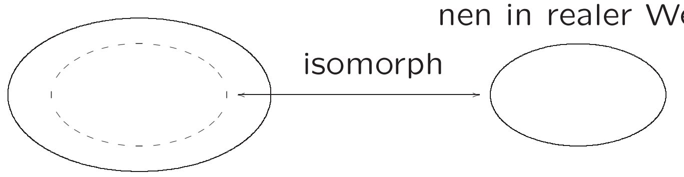
# Inhalt

# Gültige DB-Zustände (1)

- Ziel des DB-Entwurfs: Die DB sollte ein Abbild der relevanten Teilmenge der realen Welt sein.
- Aber die Definition der Grundstruktur (Entities, Attribute und Relationships) erlaubt oft zu viele (bedeutungslose, illegale) DB-Zustände:

DB-Zustände für Schema

Mögliche Situationen in realer Welt








# Gültige DB-Zustände (2)

| KUNDE | | | | |
|---|---|---|---|---|
| Kund_Nr | Name | Geb_Jahr | Stadt | ··· |
| 1 | Smith | 1936 | Pittsburgh | ··· |
| 2 | Jones | 1965 | Philadelphia | ··· |
| 3 | Brown | 64 | New York | ··· |
| 3 | Ford | 2000 | Washington | ··· |

- Kundennummern müssen eindeutig sein.
- Das Geburtsjahr muss größer als 1880 sein.
- Kunden müssen mindestens 18 Jahre alt sein.





# Gültige DB-Zustände (3)

- Zwei Fehlerarten müssen unterschieden werden:
  - ◇ Eingabe falscher Daten, d.h. der DB-Zustand entspricht zu einer anderen Situation der realen Welt als der tatsächlichen aktuellen Situation.

    Z.B. 8 Punkte für Hausaufgabe 1 in der DB vs. 10 in der realen Welt. Dann ist der DB-Zustand falsch, aber nicht das Schema. Übung: Was kann man machen, um solche Fehler zu vermeiden?
  - ◇ Eingabe von Daten, die keinen Sinn machen oder die illegal sind.

    Z.B. -5 Punkte für eine Aufgabe oder zwei Einträge für die gleiche Aufgabe und den gleichen Studenten. Wenn solche unmöglichen Daten eingegeben werden können, ist das DB-Schema falsch.





# Gültige DB-Zustände (4)

- Wenn die DB illegale oder bedeutungslose Daten enthält, wird sie mit unserem allgemeinen Verständnis der realen Welt inkonsistent.
- Wenn ein Programmierer annimmt, dass die Daten bestimmte Bedingungen erfüllen, aber sie tun das nicht, kann das seltsame Effekte haben.

  Z.B. der Programmierer nimmt an, dass keine Nullwerte auftreten (spezieller Wert für leere Tabelleneinträge, siehe Kapitel 3). Also verwendet er keine Indikatorvariable beim Abfragen der Daten. Dies funktioniert, solange keine Nullwerte auftreten. Wenn aber das Schema Nullwerte zulässt und jemand irgendwann einen Nullwert eingibt, wird das Programm abstürzen (mit einer unverständlichen Fehlermeldung).

# Integritätsbedingungen (1)

- Integritätsbedingungen ("Constraints") sind Bedingungen, die jeder DB-Zustand erfüllen muss.
- Schränkt die Menge möglicher DB-Zustände ein.

  Idealerweise zu Abbildern von Situationen der realen Welt.
- Integritätsbedingungen können als Teil des DB-Schemas definiert werden.
- Das DBMS lehnt jede Änderung ab, die eine Bedingung verletzen würde.

  Somit werden ungültige Zustände ausgeschlossen. Heutige DBMS erlauben keine beliebigen Bedingungen, nur spezielle Fälle, siehe unten.





# Integritätsbedingungen (2)

- Jedes Datenmodell hat spezielle Notation/Unterstützung für bestimmte Arten von Constraints.

  Z.B. haben im ER-Modell Schlüssel, Kardinalitätsbedingungen und die Einschränkungen bei schwachen Entities eine spezielle graphische Syntax. Diese Arten von Integritätsbedingungen werden unten erklärt.
- Werden Integritätsbedingungen benötigt, die das Datenmodell bzw. DBMS nicht unterstützt, sollte man sie dennoch beim DB-Entwurf dokumentieren.

  Z.B. als logische Formel oder in natürlicher Sprache. Man kann auch eine Anfrage schreiben, die die Verletzungen der Bedingung ausgibt (d.h. das Anfrageergebnis muss immer leer sein). Zukünftige Systeme werden wahrscheinlich auch allgemeinere Constraints unterstützen.





# Integritätsbedingungen (3)

- Erzwingung allgemeiner Integritätsbedingungen:
  - ◇ Die zur Dateneingabe verwendeten Anwendungsprogramme überprüfen die Bedingung.
  - ◇ Änderungen nur über gespeicherte Prozeduren im DBMS, die die Bedingung überprüfen.
  - ◇ DBMS führt automatisch Trigger aus (auf eine Tabelle bezogene Prozedur, die Constraints prüft), wenn Tabellendaten geändert werden.
  - ◇ Von Zeit zu Zeit wird eine Anfrage ausgeführt, die alle Verletzungen der Bedingung findet.

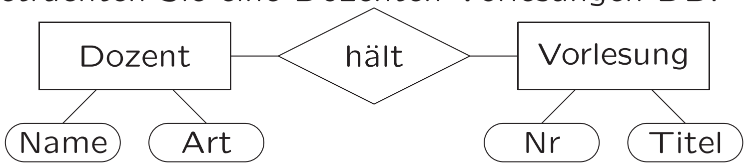




# Integritätsbedingungen (4)

Übung:

Betrachten Sie eine Dozenten-Vorlesungen-DB:




Welche der folgenden sind Integritätsbedingungen?

- Es ist legal, dass ein Dozent keine Vorlesung hält.

  ☐ Ja ☐ Nein
- Ein Dozent kann maximal 4 Vorlesungen halten.

  ☐ Ja ☐ Nein

# Integritätsbedingungen (5)

Übung, fortgesetzt:

- Das Attribut "Art" von "Dozent" kann nur die Werte 1, 2, 3, 4 haben.

  ☐ Ja ☐ Nein

- Der "Art"-Wert bedeutet folgendes: 1: Wissenschaftlicher Assistent, 2: Lehrbeauftragter, 3: Privatdozent, 4: Professor.

  ☐ Ja ☐ Nein

- Titel von Vorlesungen sollten eindeutig sein.

  ☐ Ja ☐ Nein





# Triviale/Implizierte IBen

- Eine triviale Bedingung ist eine Bedingung, die immer erfüllt ist (logisch äquivalent zu "wahr").
- Eine Bedingung $A$ impliziert eine Bedingung $B$, wenn aus $A$ ist wahr folgt, dass auch $B$ wahr ist.

  D.h. die Zustände, die $A$ erfüllen, sind eine Teilmenge der Zustände, die $B$ erfüllen. Das ist die Standarddefinition der logischen Implikation.

- Z.B. $A$: "Jeder Dozent hält 1 oder 2 Vorlesungen".
  $B$: "Dozenten können max. 4 Vorl. halten".
- Triviale/implizierte Constraints nicht angeben!

  Erhöht Komplikationen; Menge der gültigen Zustände nicht geändert!





# "Geschäftsregeln" (1)

- "Geschäftsregeln" sind Constraints sehr ähnlich: Es sind Regeln, die von Angestellten eingehalten werden müssen (um Chaos zu vermeiden).
- Constraints werden verwendet, um "Geschäftsregeln" zu implementieren.

  Gewisse Regeln, z.B. dass Kunden 18 Jahre alt sein müssen oder dass bestimmte Software nur bestimmten Kunden verkauft werden darf, können erzwungen werden, indem man solche Einträge in der DB nicht erlaubt. Ein Zustand, der die Geschäftsregeln verletzt, wird als ungültig betrachtet.





# "Geschäftsregeln" (2)

- Es gibt jedoch auch "Geschäftsregeln", die durch
  - ◇ die Grundstruktur des Modells,
  - ◇ Zugriffsrechte,
  - ◇ Anwendungsprogramme

  usw. implementiert sind.
- Geschäftsregeln sind allgemeiner als Integritätsbedingungen und existieren auch ohne Bezug zur DB.

# Zusammenfassung (1)

Warum Constraints festlegen?

- Etwas Schutz vor Eingabefehlern.
- Constraints enthalten Wissen über DB-Zustände.
- Erzwingung von Gesetzen/Unternehmensstandards
- Schutz vor Inkonsistenz bei redundant gespeicherten Daten (gleichen Daten zweimal gespeichert).
- Anfragen/Programme werden einfacher, wenn der Programmierer nicht alle Fälle beachten muss (d.h. Fälle, in denen die Constraints nicht erfüllt sind).

  Z.B. keine Indikatorvariable bei Spalten ohne Nullwerte.





# Zusammenfassung (2)

Constraints und Ausnahmen:

- Constraints können keine Ausnahmen haben.

  Jeder Versuch, ungültige Daten einzugeben, wird abgelehnt.

- Es ist eine allgemeine Erfahrung, dass es Ausnahmesituationen gibt, in denen das DBS wegen der festgelegten Constraints unflexibel erscheint.
- Nur Bedingungen, die ohne Zweifel immer gelten müssen, sollten als Constraint festgelegt werden.

  "Normalerweise"-Bedingungen könnten nützlich sein, aber nicht als Constraints. Z.B. können Programme nachfragen, wenn ein Kunde über 100 ist. Auch nützliche Information für physischen Entwurf.





# Inhalt

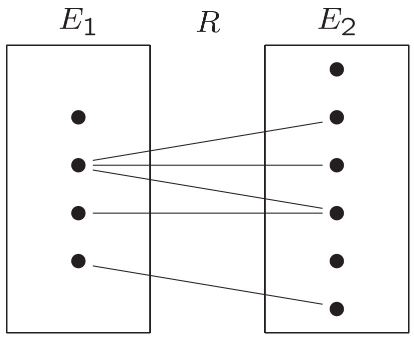






# Kardinalitäten (1)

Allgemeines Relationship:



# Kardinalitäten (2)

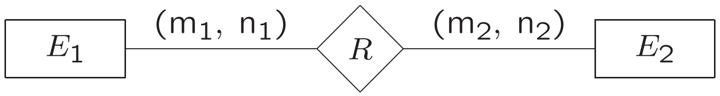
- Normalerweise gibt es keine Einschränkung, wie oft ein Entity an einem Relationship teilnimmt.
- Ein Entity kann mit einem, mit mehreren oder mit keinem Entity eines anderen Typs verbunden sein.
- Oft weiß man jedoch, wie viele $E_2$-Entities zu einem $E_1$-Entity in Beziehung stehen.






# Kardinalitäten (3)

- In der (min,max)-Notation spezifiziert man ein Intervall für die Anzahl der ausgehenden Kanten jedes Entities:




- D.h. ein Entity des Typs $E_1$ kann zu $m_1$ bis $n_1$ Entities des Typs $E_2$ in Beziehung stehen.
- Entsprechend ist $m_2$ die kleinste (und $n_2$ die größte) Anzahl von $E_1$-Entities, zu denen ein $E_2$-Entity in Beziehung stehen muss (darf).





# Kardinalitäten (4)

Formale Semantik:

- Für jeden DB-Zustand $\mathcal{I}$ u. jedes Entity $e_1 \in \mathcal{I}(E_1)$:
  $$m_1 \le |\{e_2 \in \mathcal{I}(E_2) \mid (e_1, e_2) \in \mathcal{I}(R)\}| \le n_1.$$
- Für jeden DB-Zustand $\mathcal{I}$ u. jedes Entity $e_2 \in \mathcal{I}(E_2)$:
  $$m_2 \le |\{e_1 \in \mathcal{I}(E_1) \mid (e_1, e_2) \in \mathcal{I}(R)\}| \le n_2.$$

Erweiterung:

- $*$ als Maximum bedeutet, dass es kein Limit gibt.
- $(0, *)$ bewirkt also gar keine Einschränkung.
  Diese Kardinalität ist der Default (wenn keine Kardinalität festgelegt).





# Kardinalitäten (5)

Beispiel:

- Ein Mann ist mit maximal einer Frau verheiratet und umgekehrt:


- Ein Flughafen liegt in genau einem Land, ein Land kann mehrere Flughäfen haben (oder gar keinen):

# Kardinalitäten (6)

Übung: Kardinalitäten festlegen.

- Hinweis: Neben normalen Kunden enthält die DB auch solche, die noch nichts bestellt haben, sondern nur einen Produktkatalog erhalten haben.


- Eine Bestellung kann mehrere Produkte umfassen:






# Kardinalitäten festlegen (1)

- Machen Sie ein Beispiel eines gültigen DB-Zustands: Einige Entities des Typs $E_1$, einige des Typs $E_2$ und Kanten zwischen zusammengehörigen Entities.
- Nun zählt man die ausgehenden Kanten für alle Entities des Typs $E_1$. (Später das gleiche mit $E_2$.)
- Z.B. gibt es im Relationship auf Folie 8-56 $E_1$-Entities mit 0, 1 und 3 ausgehenden Kanten.
- Also ist $(0, 3)$ die stärkste mögliche Kardinalitätsbedingung (Einschränkung).

  Die Bedingung muß ja im als gültig angenommenen Zustand gelten.





# Kardinalitäten festlegen (2)

- Kenntnis der Anwendung kann zu schwächeren Einschränkungen führen (d.h. größere Intervalle), z.B. $(0, 5)$ oder sogar $(0, *)$.

  $(a, b)$ ist schwächer als $(c, d)$, wenn $a \leq c$ und $d \leq b$.

- Die Kardinalitäten $(0, 1)$, $(1, 1)$ und $(0, *)$ sind besonders üblich und im relationalen Schema leicht zu erzwingen.

  Z.B. statt $(0, 30)$ wählt man lieber $(0, *)$ (leichter). Wenn jedoch 30 eine harte Grenze ist (d.h. der DB-Zustand wäre ungültig, wenn dies verletzt ist), sollte man die Kardinalität $(0, 30)$ wählen und diese über Constraints, Trigger oder Anwendungsprogramme erzwingen.





# Übliche Fälle (1)

- Die Minimalkardinalität wird normalerweise 0 oder 1 sein und die Maximalkardinalität 1 oder $*$.

  Also sind nur $(0, 1)$, $(1, 1)$, $(0, *)$, $(1, *)$ in der Praxis üblich. Davon ist jedoch $(1, *)$ im relationalen Schema schwer umzusetzen.

- Um das Relationship zu verstehen, muss man die Kardinalitäten auf beiden Seiten kennen.
- Aufgrund der Maximalkardinalitäten unterscheidet man "viele zu viele" (n:m), "eins zu viele" / "viele zu eins" (1:n) und "eins zu eins" (1:1)-Relationships.

# Übliche Fälle (2)

"Viele zu Viele"-Relationships:

- Beide Maximalkardinalitäten sind "$*$":

  Minimal-Kardinalitäten können 0 oder 1 sein, siehe Folie 8-70.


- Ein Student kann viele Vorlesungen besuchen und eine Vorlesung kann viele Studenten haben.
- Das ist der allgemeinste Fall.
- Bei der Übersetzung ins relationale Modell benötigen "viele zu viele"-Relationships eine extra Tablle.





# Übliche Fälle (3)

"Eins zu Viele"-Relationships:

- Maximalkardinalität 1 auf der "viele"-Seite und $*$ auf der "eins"-Seite.


- Z.B. ein Dozent hält viele Vorlesungen, aber jede Vorlesung hat maximal einen Dozenten ("ein Dozent, viele Vorlesungen").

  Also ist das "eins zu viele" von Dozent zu Vorlesung.





# Übliche Fälle (4)

"Eins zu Viele"-Relationships, fortgesetzt:

- Beachte: 1/eins und $*$/viele sind vertauscht!
  - ◇ "Dozent" ist die "eins"-Seite, aber hat Maximalkardinalität $*$.
  - ◇ "Vorlesung" ist die "viele"-Seite, aber hat Maximalkardinalität 1.
- "Eins zu viele"-Relationships brauchen keine extra Tabelle bei der Übersetzung ins relationale Modell.

  Das sind wahrscheinlich die üblichsten Relationships.

- "Viele zu eins": symmetrisch (z.B. "gehalten von")





# Übliche Fälle (5)

"Eins zu Eins"-Relationships:

- Maximalkardinalität 1 auf beiden Seiten:


- Ein Mann kann mit maximal einer Frau verheiratet sein, eine Frau mit maximal einem Mann.

  Oder: "Ist Chef von" zwischen Angestellten und Abteilungen: Jede Abteilung hat genau einen Chef (notwendige Teilnahme) Angestellte kann maximal eine Abteilung führen (normale Angestellte führen keine Abteilung: optionale Teilnahme).

# Übliche Fälle (6)

"Eins zu Eins"-Relationships, Forts.:

- Eins-zu-eins Beziehungen sind in der Praxis recht selten.
- Wenn Sie in einem DB-Schema eine eins-zu-eins Beziehung verwenden wollen, prüfen Sie genau, ob es sich nicht doch um eine eins-zu-viele Beziehung handelt.

  Natürlich gibt es eins-zu-eins Beziehungen (wie "verheiratet mit"). Aber schon bei "Angestellter ist Chef von Abteilung" ist nicht so klar, ob in Ausnahmefällen nicht ein Angesteller vorübergehend auch zwei Abteilungen leiten kann. In Klausuren sind Fehler (in viel klareren Fällen) nicht selten.





# Übliche Fälle (7)

Teilnahme:

- Die Minimalkardinalitäten legen die Teilnahme von Entities in einem Relationship fest:
  - ◇ total (zwingend): Jedes Entity muss am Relationship beteiligt sein.
  - ◇ partiell (optional): Einige Entities sind am Relationship beteiligt, andere nicht.
- Minimalkardinalitäten sind für die Klassifikation des Relationships in "viele zu viele", "eins zu viele" oder "eins zu eins" nicht wichtig.





# Übliche Fälle (8)

- Im folgenden Beispiel
  - ◇ ist die Teilnahme von "Vorlesung" zwingend (jede Vorlesung muss einen Dozenten haben),

    Das könnte in der Praxis zu einschränkend sein. Es wird am Semesteranfang gelten, aber nicht während der Planung.
  - ◇ ist die Teilnahme von "Dozent" optional (Dozenten können Forschungssemester haben).

    D.h. es kann Dozenten geben, die im laufenden Semester keine Vorlesungen halten.






# Spezifikation mit Logik

- Maximalkardinalität 1: Jede Vorlesung wird von maximal einem Dozenten gehalten:

  $$\forall\,\texttt{Vorlesung V},\ \texttt{Dozent D1},\ \texttt{Dozent D2}:\ \texttt{hält}(\texttt{D1},\texttt{V}) \wedge \texttt{hält}(\texttt{D2},\texttt{V}) \rightarrow \texttt{D1} = \texttt{D2}.$$
- Maximalkardinalität $*$: Das ist kein Constraint.
- Minimalkardinalität 1: Jede Vorlesung wird von mindestens einem Dozenten gehalten:

  $$\forall\,\texttt{Vorlesung V}:\ \exists\,\texttt{Dozent D}:\ \texttt{hält}(\texttt{D},\texttt{V})$$
- Minimalkardinalität 0: Das ist kein Constraint.

# Alternative Notationen (1)

- Es gibt viele Varianten der ER-Notation. Kardinalitäten sind ein Punkt, in dem sie sich stark unterscheiden.
- Z.B. kann man nur “viele zu viele” (N:M), “eins zu viele” (1:N) und “eins zu eins” (1:1) verwenden.


- Die Teilnahme (zwingend/optional) wird oft nicht spezifiziert (manchmal: doppelte Linie: zwingend).





# Alternative Notationen (2)

- Oracle Designer hat die “Krähenfuß”-Notation:

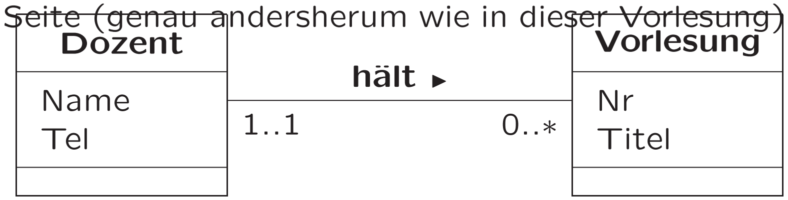

Der “Krähenfuß” stellt die “viele”-Seite dar. Gestrichelte Linie bedeutet optionale Teilnahme, durchgezogene Linie totale Teilnahme. Man beachte, dass die Minimalkardinalität auf der gleichen Seite steht wie in unserer Notation, wohingegen die Maximalkardinalität auf der anderen Seite steht. Relationships haben zwei Namen, einer in jede Richtung (“Rollennamen”). Attribute stehen in der Entity-Box. Das Symbol “#” bedeutet Primärschlüsselattribut (eindeutige Identifikation, siehe unten), Symbol “*” immer definiertes Attribut (“NOT NULL”) und Symbol “○” optionales Attribut (kann NULL sein).





# Alternative Notationen (3)

- Bei UML Klassendiagrammen stehen Minimal- und Maximal-Kardinalitäten auf der gegenüberliegenden Seite (genau andersherum wie in dieser Vorlesung):




Man kann `1..1` zu `1` abkürzen, und `0..*` zu `*`. Der dritte Abschnitt im Rechteck für die Klassen enthält die Methoden (bei Datenbanken eher selten). Es gibt keine in UML eingebaute Notation für Schlüssel, man kann aber Erweiterungsmechanismen verwenden, z.B. “{pk}” hinter die Attribute des Primärschlüssels schreiben (kein Standard).





# Inhalt

1. Überblick über den Datenbank-Entwurf
2. Grundlegende ER-Elemente
3. Integritätsbedingungen: Allg. Bemerkungen
4. Relationship-Arten (Kardinalitäten)
5. Schlüssel, schwache Entities
6. Qualität eines ER-Schemas

# Schlüssel (1)

- Ein Schlüssel eines Entity-Typs $E$ ist ein Attribut, das die Entities dieses Typs eindeutig identifiziert.
- Es darf nie zwei Entities geben, die den gleichen Wert für das Schlüsselattribut haben.
- Z.B. ist in Amerika die Sozialversicherungsnummer (SSN) ein Schlüssel für Personen: Zwei verschiedene Personen haben nie die gleiche SSN.

  Ich habe gehört, dass in sehr seltenen Fällen zwei Personen die gleiche SSN haben. Wenn das stimmt, ist die SSN kein Schlüssel.





# Schlüssel (2)

- Man kann auch Kombinationen von zwei oder mehr Attributen als Schlüssel deklarieren.

  Dann dürfen zwei Entities nicht gleichzeitig in jedem dieser Attribute übereinstimmen.
- Z.B. kann man Vorname und Nachname zusammen als Schlüssel für Dozenten verwenden.
- Dann können zwei Dozenten den gleichem Nachnamen haben, solange sich ihre Vornamen unterscheiden.

  Auch möglich: zwei Dozenten mit gleichem Vornamen und unterschiedlichen Nachnamen.

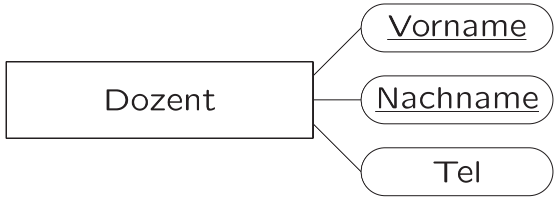




# Schlüssel (3)

Graphische Syntax:

- Schlüssel werden in ER-Diagrammen markiert, indem man die Schlüsselattribute unterstreicht:




- Nur Entity-Typen können Schlüsselattribute haben.

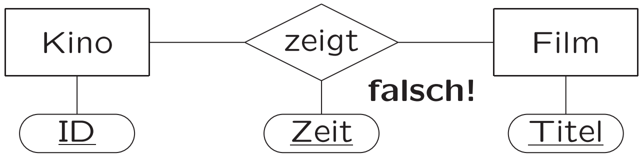
  Schlüssel können nicht für Relationships deklariert werden (aber Kardinalitäten sind so ähnlich wie Schlüssel für Relationships).





# Schlüssel (4)

- Das in der Wirtschaftsinformatik verwendete Lehrbuch erlaubt, Attribute von Relationships zu unterstreichen. Das wird hier als Fehler gewertet:




  Begründung: So ist der Schlüssel keine Integritätsbedingung mehr. Es werden keine Zustände eingeschränkt, sondern die Semantik des Relationships verändert: Es ist keine Menge von Paaren von Entities mehr. Das ist zu relational gedacht (man braucht später natürlich eine Tabelle mit allen drei Attributen als Schlüssel). Unseres Wissens ist das Lehrbuch das einzige, was dieses Konstrukt erlaubt.

# Schlüssel (5)

Spezifikation mit Logik:

- Schlüsselbedingungen können als logische Formeln spezifiziert werden. Z.B. bedeutet der Schlüssel "Vorname, Nachname" von "Dozent":

$$\forall\, \texttt{Dozent X, Dozent Y:}$$
$$\texttt{X.Vorname} = \texttt{Y.Vorname} \land$$
$$\texttt{X.Nachname} = \texttt{Y.Nachname} \rightarrow \texttt{X} = \texttt{Y.}$$

- Äquivalente Formulierung:

$$\neg\exists\, \texttt{Dozent X, Dozent Y:}$$
$$\texttt{X.Vorname} = \texttt{Y.Vorname} \land$$
$$\texttt{X.Nachname} = \texttt{Y.Nachname} \land \texttt{X} \neq \texttt{Y.}$$





# Schlüssel (6)

Implikation von Schlüsselbedingungen:

- Schlüsselbedingungen werden schwächer (mehr Zustände gültig), wenn Attribute zugefügt werden.
- Z.B. betrachten wir die Professoren Stefan Posch, Nina Brass, Stefan Brass. Dieser Zustand
  - ◇ verletzt die Schlüsselbedingung für Vorname,

    Es gibt zwei "Stefans".
  - ◇ verletzt die Schlüsselbedingung für Nachname,
  - ◇ erfüllt die Schlüsselbedingung für die Kombination von Vorname, Nachname.





# Schlüssel (7)

Minimalität von Schlüsseln:

- Wenn ein Schlüssel deklariert wurde, z.B. PersNr, dann ist jede Obermenge (z.B. PersNr, Nachname) automatische eine eindeutige Identifizierung.

  Wenn es keine zwei Professoren mit der gleichen Personalnummer geben kann, dann gibt es natürlich auch keine zwei Professoren, die die gleiche PersNr und den gleichen Nachnamen haben.
- Es ist üblich, "Schlüssels" so zu definieren, dass die Menge der Schlüsselattribute $\{A_1, \ldots, A_k\}$ minimal sein muss (zusätzlich zur Eindeutigkeitsbedingung).

  Da die eindeutige Identifizierung automatisch für Obermengen gilt, sind "nichtminimale Schlüssel" tatsächlich nicht interessant.





# Schlüssel (8)

Minimalität von Schlüsseln, fortgesetzt:

- Minimalität bedeutet, dass man kein Schlüsselattribut weglassen kann, ohne die Eigenschaft der eindeutigen Identifikation zu verlieren.
- Mit dieser Definition ist ein Schlüssel nicht nur ein
  - ◇ Constraint, der ungültige Zustände ausschließt,
  - ◇ sondern auch eine Aussage über die reale Welt, dass bestimmte Zustände möglich sind.
- In der Literatur wird ein Schlüssel, der nicht minimal ist, "Superkey" genannt.

# Mehrere Schlüssel (1)

- Ein Entity-Typ kann mehrere Schlüssel haben:
  - ◇ Z.B. ist PersNr ein Schlüssel.
  - ◇ Die Kombination von Vorname und Nachname ist ein weiterer Schlüssel.
- Beide Schlüssel sind minimal, weil keiner eine Obermenge des anderen ist. (Keiner der beiden impliziert den anderen.)

  Es ist egal, dass der erste Schlüssel nur ein Attribut hat und der zweite aus zwei Attributen besteht.





# Mehrere Schlüssel (2)

- Einer der Schlüssel wird als “Primärschlüssel” deklariert. Andere Schlüssel werden “Alternativ- oder Sekundärschlüssel” genannt.

  SQL verwendet die Begriffe `PRIMARY KEY` für Primärschlüssel und `UNIQUE` für Sekundärschlüssel.
- Der Primärschlüssel sollte ein Schlüssel sein, der nur aus einem Attribut besteht und wenn möglich nie geändert wird.

  Nach der Übersetzung ins relationale Modell wird der Primärschlüssel in anderen Tabellen verwendet, die sich auf Entities dieses Typs beziehen. In manchen Systemen kann der Zugriff über den Primärschlüssel sehr schnell sein. Ansonsten ist die Wahl des Primärschlüssels egal.

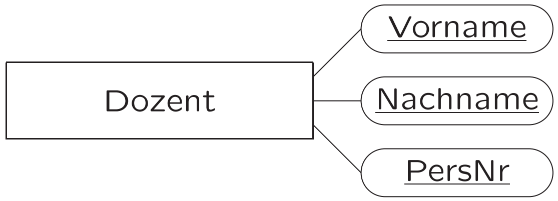




# Mehrere Schlüssel (3)

- Die graphische Syntax erlaubt für jeden Entitytyp nur die Angabe eines Schlüssels (Primärschlüssel).

  In CASE-Tools kann man auch Sekundärschlüssel speichern. Ansonsten sollte man die Sekundärschlüssel in Prosa erwähnen.
- Dieses Beispiel würde bedeuten, dass es nur einen Schlüssel gibt, der aus allen drei Attributen besteht:








# Sind Schlüssel notwendig?

- Das ER-Modell verlangt nicht die Definition eines Schlüssels für jeden Entity-Typ (Objektidentität).
- Bei der Übersetzung ins relationale Modell benötigt man jedoch für jeden Entity-Typ einen Schlüssel.

  Bei “schwachen Entity-Typen” (siehe unten) enthält der “Schlüssel” ein Relationship.
- Gibt es keine natürlichen Schlüssel, verwendet man Zahlen zur Identifizierung (“künstliche Schlüssel”).
- CASE-Tools wie Oracle Designer machen das automatisch, wenn kein Schlüssel benannt wurde.

# Künstliche Schlüssel (1)

- Künstliche Schlüssel sind einfach. Es gibt aber auch Nachteile, wenn man sie verwendet.

  Dinge wie "Bestellnummer", die schon in der realen Welt verwendet werden, sind an der Grenze zwischen natürlich und künstlich.

- Die Wahl eines natürlichen Schlüssels (keine künstliche Identifikationsnummer) ist ziemlich schwierig.

  Eines von Murphys Gesetzen: Es gibt immer Ausnahmen.

- Oft hilft das Nachdenken über Schlüssel, die wahre Bedeutung des Konzepts besser zu verstehen.

  Z.B. angebotene Vorlesung eines Semesters vs. Eintrag in einem Vorlesungskatalog.





# Künstliche Schlüssel (2)

- Auch bei Verwendung künstlicher Zahlen sollte man über den Identifikationsmechanismus nachdenken, der in Anwendungsprogrammen verwendet wird.

  Es ist gefährlich, wenn verschiedene Anwendungsprogramme verschiedene Identifikationsmechanismen für die gleiche Sache verwenden.

- Wenn die Nichtpräsenz in der DB verwendet wird, um Aussagen über andere Objekte der realen Welt zu machen (z.B. "Schufa"), müssen all diese Objekte eindeutig identifizierbar sein.

  Die Objekte, über die Daten gespeichert sind, könnten den Schlüssel verletzen, obwohl die Teilmenge in der DB den Schlüssel erfüllt.





# Künstliche Schlüssel (3)

- Der Zweck von Schlüsseln ist nicht nur die Identifikation von Entities.
- Schlüssel sollten auch helfen, Duplikate in der DB zu vermeiden. Künstliche Schlüssel tun dies nicht.
- Oft sind Duplikate keine exakten Kopien: Z.B. andere Abkürzungen oder Großschreibung verwendet.

  Dann hilft das Konzept eines Schlüssels nicht. Darüberhinaus wird kein Constraint benötigt — eine Warnung würde genügen.

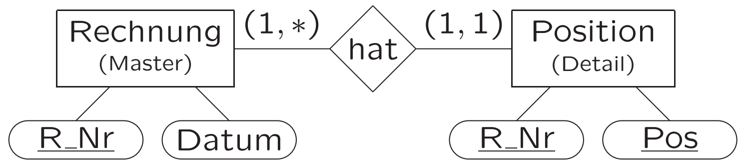
- Man sollte vor der Programmierung über Möglichkeiten zur Duplikatvermeidung nachdenken.





# Schwache Entities (1)

- Ein Entity kann eine Art "Detail" beschreiben, das ohne "Master-" Entity nicht existieren kann.
- Dann gibt es ein Relationship mit einer (1,1)-Kardinalität, die zum Master (Elternteil) zeigt.
- Außerdem wird der Schlüssel des Master-Entities ein Teil des Schlüssels des Detail-Entities:



# Schwache Entities (2)

- Ohne spezielle Konstrukte für diese Situation würde man folgenden Constraint benötigen:
  - ◇ Wenn zwei Entities mit “hat” verbunden sind,
  - ◇ dann stimmen ihre Attribute “R_Nr” im Wert überein.

    Z.B. kann Rechnung 12 nicht Pos. 2 in Rechnung 6 als Detail haben.
- Solche Constraints tauchen auf, wenn ein Entity selbst keinen vollständigen Schlüssel hat, sondern nur eindeutig in Bezug auf ein anderes Entity ist.

  D.h. es muss ein Schlüsselattribut des zugehörigen Entities “leihen”.





# Schwache Entities (3)

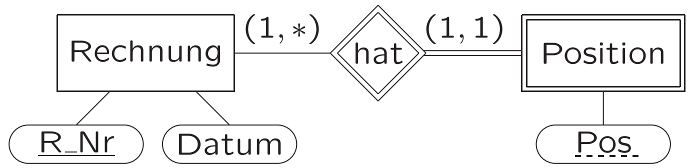
- In solchen Fällen gibt es immer zusammengesetzte Schlüssel:
  - ◇ Ein Klassenraum wird durch ein Gebäude und durch eine Zimmernummer identifiziert.
  - ◇ Eine Unteraufgabe wird durch eine Aufgabennummer (z.B. 1) und einen Buchstaben (z.B. a) identifiziert.
  - ◇ Eine Web-Seite wird durch einen Web-Server und einen Pfad auf diesem Server identifiziert.





# Schwache Entities (4)

- Zusätzlich besteht eine Existenzabhängigkeit: Wird ein Gebäude abgerissen, verschwinden die Räume darin automatisch.
- Schwache Entities haben alle diese Eigenschaften.
- Sie sind durch doppelte Linien an Rechteck, Verbindungslinie und Raute gekennzeichnet:








# Schwache Entities (5)

- Nur die Erweiterung des Schlüssels wird dargestellt. Da es nur ein Teil-Schlüssel ist, wird er gestrichelt unterstrichen.
- Also besteht der echte Schlüssel des schwachen Entities aus den Schlüsselattributen des Master-Entities (automatisch vererbt) und dem gestrichelt unterstrichenen Teil-Schlüssel.
- Diese spezielle Schlüsselkonstruktion ist das wesentliche Kennzeichen eines schwachen Entities.

  Nicht die Existenzabhängigkeit! Die besteht im Prinzip immer bei der minimalen Kardinalität 1 (automatisches Löschen ist andere Frage).

# Schwache Entities (6)

- Man kann es auch so verstehen, dass hier das Relationship zur Identifikation des Entities beiträgt, wohingegen normalerweise nur Attribute für diesen Zweck verwendet werden (als Schlüssel).

  Die doppelte Linie kann als "unterstrichene Linie" gesehen werden.

- Die Kardinalität $(1, 1)$ der Beziehung zum Master-Entity wird bei schwachen Entities impliziert (muss nicht explizit angegeben werden).

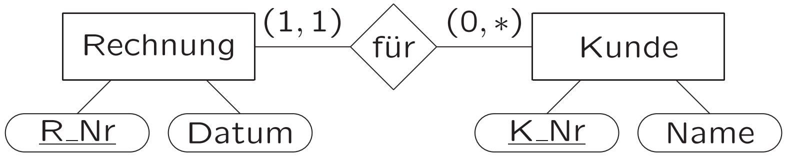
  Andere Relationships können nicht verwendet werden: Es würde kein eindeutiges Master-Entity geben, von dem der Schlüsselwert geerbt werden kann.





# Schwache Entities (7)

- Eine $(1, 1)$-Kardinalität bedeutet nicht unbedingt "schwaches Entity". Das Entity kann trotzdem einen eigenen Schlüssel haben:




- Ein schwaches Entity wird nur benötigt, wenn der Schlüssel des Entities den Schlüssel eines anderen enthält.





# Schwache Entities (8)

- Ein schwaches Entity muss weitere Schlüsselattribute zu den geerbten hinzufügen.

  Wenn die Schlüssel der beiden Entity-Typen die gleichen wären, sollte eine Spezialisierung verwendet werden (→ Vorlesung über DB-Entwurf).

- Manchmal wird das Master-Entity "Eltern-Entity" und das abhängige schwache Entity "Kind-Entity" genannt.
- Entities mit eigenem Schlüssel (nicht-schwache Entities) werden reguläre/starke Entities genannt.





# Schwache Entities (9)

- Schwache Entities sind normale Entities, außer dass ihr Schlüssel auf spezielle Art gebildet wird.
- Somit können sie auch normale weitere Relationships haben:


- Schwache Entities können selbst Master-Entities für andere schwache Entities sein.

  Es kann eine ganze Hierarchie von Master-Detail-Relationships geben. Aber Zyklen sind verboten.

# Schwache Entities (10)

Übung:

- Modellieren Sie Tests für ein E-Learning Angebot im Web (mehrere Multiple-Choice-Tests).
- Jeder Test wird durch einen Titel, jede Frage in einem Test durch eine Zahl und jede Antwort zu einer Frage durch einen Buchstaben identifiziert.

  Für jede Frage und Antwort muss der Text gespeichert werden und Antworten müssen in richtige und falsche klassifiziert werden.
- Wie lauten die vollständigen Schlüssel (inklusive "geborgter" Attribute) der drei Entity-Typen?





# Association-Entities (1)

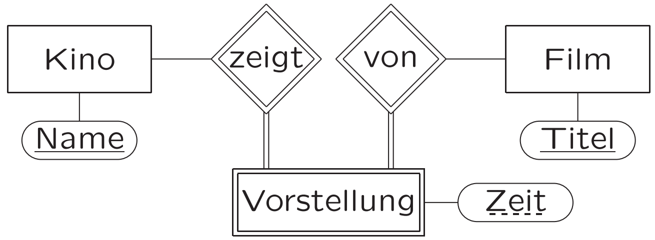
- Manchmal ist es notwendig, ein Relationship in ein Entity umzuwandeln, z.B. weil:
  - ◇ Ternäre Relationships hier ausgeschlossen sind.
  - ◇ Viele CASE-Tools Relationship-Attribute nicht unterstützen.
  - ◇ Manche vorschlagen, viele-zu-viele-Relationships auf diese Art durch Entities zu ersetzen.
  - ◇ Ein Relationship zwischen einem Relationship und einem Entity-Typ benötigt wird.
  - ◇ Es mehrwertige Relationship-Attribute gibt.





# Association-Entities (2)

- Das Relationship wird dann zu einem schwachen Entity. Es erbt die Schlüssel der Entity-Typen, die am ursprünglichen Relationship beteiligt sind.
- Schwache Entities können mehrere Master/Eltern haben:







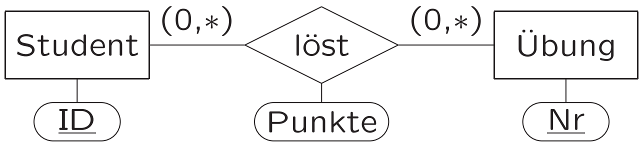

# Association-Entities (3)

- Im obigen Beispiel besteht der Schlüssel des schwachen Entities "Vorstellung" aus
  - ◇ "Name" (geerbt von "Kino"),
  - ◇ "Titel" (geerbt von "Film") und
  - ◇ "Zeit".
- Schwache Entity-Typen mit mehreren Mastern werden manchmal "Association-Entities" genannt.

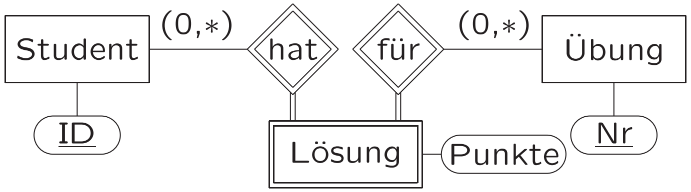
  Wenn zwei Eltern Schlüsselattribute mit gleichem Namen haben, muss (mindestens) eins implizit umbenannt werden.
- "Schwaches Entity" ist jedoch genauso gut.

# Association-Entities (4)

- Relationship:




- Association-Entity (vollkommen äquivalent):








# Graphische Syntax (1)

- Alle Regeln der Folien 8-24 bis 8-26 gelten noch.
- Name von Ovalen (mit Kästen verbunden) darf
  - ◇ unterstrichen (bei Kasten mit einfachem Rand)
  - ◇ oder gestrichelt unterstrichen sein (wenn der Kasten einen doppelten Rand hat).

  Das sind die einzigen Fälle, in denen Unterstreichen erlaubt ist.
- Linien zwischen Kästen und Rauten können mit einem Zahlenpaar $(n, m)$ versehen sein, wobei $m$ “$*$” sein darf. Wenn $m$ nicht $*$ ist, muss $n \leq m$ gelten.

  “$*$” nicht erlaubt an 1. Stelle, “0” macht an 2. Stelle keinen Sinn.





# Graphische Syntax (2)

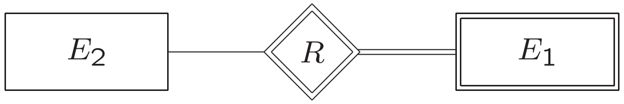
- Kästen können einen doppelten Rand haben. In diesem Fall muss mindestens eine Linie zu einer Raute auch doppelt sein.
- Nur Linien zwischen Kästen und Rauten dürfen verdoppelt sein. In diesem Fall müssen Raute und Kasten einen doppelten Rand haben. Die Linie auf der anderen Seite der Raute muss einfach sein.
- Wenn eine doppelte Linie mit einem Zahlenpaar versehen ist, muss dies $(1, 1)$ sein.





# Graphische Syntax (3)

- Betrachten Sie einen Graphen mit
  - ◇ allen Kästen mit doppeltem Rand als Knoten,
  - ◇ eine Kante von $E_1$ zu $E_2$, wenn $E_1$ durch eine doppelte Linie mit einer Raute verbunden ist, die auch mit $E_2$ verbunden ist:




- Dieser Graph darf keine Zyklen enthalten.

  D.h. ein schwacher Entity-Typ kann nicht direkt oder indirekt sein eigener Master sein.

# Inhalt







# Geeignete Namen (1)

- Namen sollten selbstdokumentierend sein.

  Der Zusammenhang zur realen Welt muss klar sein. Wenn nötig, fügt man Kommentare, Erklärungen oder Beispieldaten hinzu.

- Namen sollten nicht zu lang sein.

  Namen mit mehr als 15–20 Buchstaben werden unhandlich.

- Die Wahl guter Namen verlangt einige Überlegung. Aber die investierte Zeit wird sich später auszahlen.

  Es kann helfen, die Namen mit anderen Leuten zu diskutieren.

- Man sollte immer versuchen, konsistent zu bleiben!

  Abkürzungen konsistent verwenden. "_" konsistent verwenden. In einem Team sollten alle den gleichen Stil verwenden.





# Geeignete Namen (2)

Entity-Typen:

- Meist Substantive für Entity-Typen verwendet.
- Ich bevorzuge die Singularform (einzelne Entities).

  Manche Autoren verwenden Pluralformen. Nach der Übersetzung ins relationale Modell sind diese Namen wohl passender (Wenn man aber an die Deklaration von Tupelvariablen in SQL denkt, passt die Singularform besser.) Oracle Designer nimmt den Singular für Entity-Typen und den Plural für die dazugehörigen Tabellen. Wichtig ist nur, dass man konsistent einen Stil verwendet.

- Keine Namen wie "Kundendaten" verwenden: Das Ziel ist ein Modell von der realen Welt, nicht von der Datenbank.





# Geeignete Namen (3)

Relationships:

- Oft Verben für Relationship-Namen verwendet.
- Relationship-Namen (z.B. "hält") sollten sich von links nach rechts und von oben nach unten lesen.

  In vielen ER-Modell-Varianten haben Relationships zwei Namen: Einer in jede Richtung. Das löst das Leseproblem.

- Präpositionen wie "von" sind recht unspezifisch. Man kann sie jedoch mit anderen Worten kombinieren, um sie klarer zu machen (z.B. "Dozent von").

  Manche verwenden auch Substantive wie "Dozenteneinteilung".

# Attribut vs. Entity (1)

- Manchmal nicht klar, ob neuer Entity-Typ benötigt wird:


- Man könnte den Namen des Dozenten als Attribut für Vorlesung modellieren:






# Attribut vs. Entity (2)

- Vorteile der Variante mit dem "Dozent"-Entity:
  - ◇ Wenn später mehr Daten über Dozenten gespeichert werden sollen (z.B. Tel.-Nummer), muss das Schema nur wenig geändert werden.
  - ◇ Es ist unwahrscheinlicher, dass der gleiche Dozent in verschiedenen Schreibweisen auftaucht (Tippfehler).

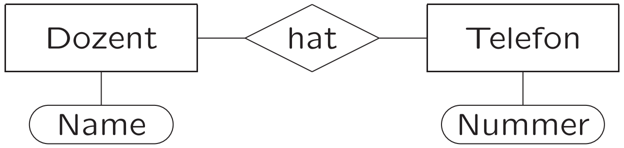
    Natürlich hängt das auch von den Anwendungsprogrammen ab.
- Aber die Lösung mit "DozName" als Attribut ist einfacher.





# Attribut vs. Entity (3)

- Im Prinzip kann jedes Attribut in einen eigenen Entity-Typ umgewandelt werden:



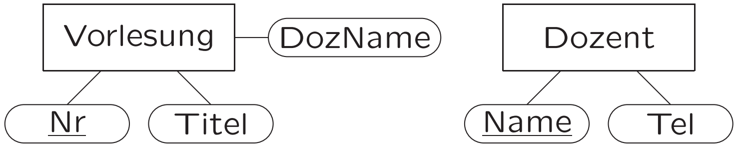

- Dieser Entwurf wäre nur für die Telefongesellschaft interessant. Ansonsten ist er zu kompliziert.
- Man vermeide Entity-Typen, die eigentlich nur Datentypwerte sind.





# Attribut vs. Relationship

- Im Prinzip kann ein Relationship durch Attribute mit gleichem Wert dargestellt werden:




- Das ist im ER-Modell nicht richtig. Relationships sollten explizit dargestellt werden.

  Im relationalen Modell wäre es richtig. Dort gibt es kein explizites "Relationship"-Konstrukt. Aber solche "Fremdschlüssel" in ER-Schemata zu verwenden, ist wirklich schlecht.

# Redundante Information (1)

- Redundante Informationen in einem ER-Schema sind schlecht: Sie verkomplizieren das Schema und die Anwendungs-Programmierung, sowie ggf. später notwendige Änderungen.

  Man braucht mindestens eine Integritätsbedingung, die sicherstellt, dass beide Kopien der Information konsistent bleiben.

- Redundante Information kann im physischen Entwurf aus Performancegründen eingeführt werden.
- Außerdem kann man später Sichten definieren, die abgeleitete Informationen enthalten.





# Redundante Information (2)

- Man sollte Attribute, die über ein Relationship erreicht werden können, nicht wiederholen:


- Viele Studenten, die bereits Erfahrung im relationalen DB-Entwurf haben, machen diesen Fehler.

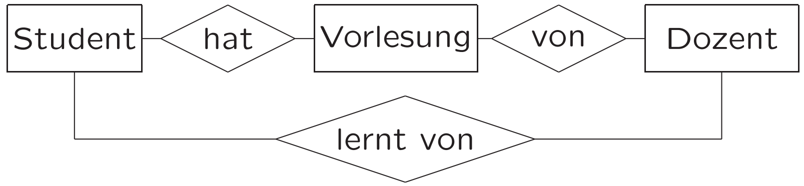
  Es wird ein solche Spalte nach der Übersetzung ins relationale Modell geben (siehe unten). Aber es gibt keine Relationships im relationalen Modell. Ein Relationship <u>und</u> ein solches Attribut zu haben, ist redundant.





# Redundante Information (3)

- Es ist falsch, die Komposition zweier Relationships wieder als Relationship zu definieren:




- Solche Zyklen in einem Diagramm sollten genau auf mögliche Redundanzen untersucht werden.

  Natürlich sind Zyklen nicht immer redundant oder schlecht, aber es existiert oft eine Abhängigkeit (→ Integritätsbedingung).