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 Anwendungsexperten (z.B. spätere Nutzer der Datenbank). Diese erwähnen aber eventuell Dinge nicht, die für sie selbstverständlich sind.
  - ◇ 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.

ER-Modell: Einordnung (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 Unterscheidung wird nicht benötigt, um dem Informationsbedarf der Anwendungen zu genügen. Aber es verdeutlicht den Zusammenhang zwischen dem Schema und der realen Welt.

ER-Modell: Einordnung (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.
- Viele Variationen / Erweiterungen des ER-Modells wurden vorgeschlagen.

  Z.B. verwenden wir hier die Notation von Barker (Buch 1989/90). Diese enthält u.a. auch Subklassen, die wir aber erst in "Datenbanken IIA" behandeln. Die leichte Erweiterbarkeit (weil nicht an DBMS gebunden) ist auch ein Vorteil des ER-Models.

ER-Modell: Einordnung (3)

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

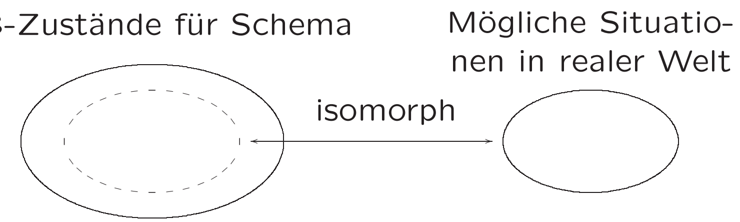
  Z.B. sind Oracle Designer und Oracle SQL Developer Data Modeler CASE-Tools für DB-Anwendungen, mit denen man u.a. ER-Diagramme zeichnen kann. (CASE: Computer-Aided Software Engineering.) Alternativen: Sybase PowerDesigner, CA ERwin, . . .
- Das ER-Modell ist ein Standard-Tool für den konzeptionellen 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

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:



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: Welche der folgenden Aussagen zu einer Dozenten-Vorlesungs-DB sind Integritätsbedingungen?

- Es ist legal, dass ein Dozent keine Vorlesung hält.
- Ein Dozent kann maximal 4 Vorlesungen halten.
- Die Spalte "Art" der Tabelle "Dozent" kann nur die Werte 1, 2, 3, 4 haben.
- Der "Art"-Wert bedeutet folgendes: 1: Assistent, 2: Lehrbeauftragter, 3: Privatdozent, 4: Professor.
- Titel von Vorlesungen sollten eindeutig sein.

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

Beispiel (1)

ER-Diagramm in Barker-Notation:

- Diese Miniwelt enthält Dozenten und Vorlesungen.

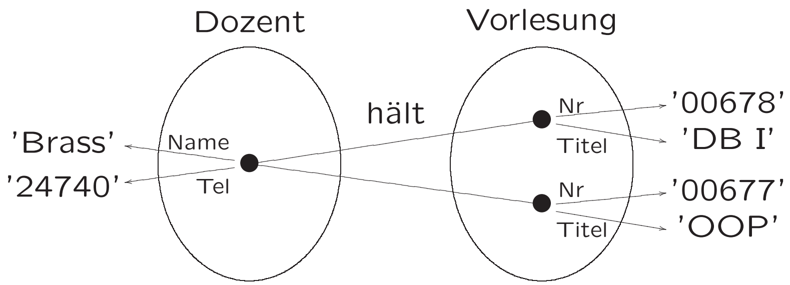
  "Entity-Typen", entspricht Klassen von Objekten. Über die Objekte sind weitere Daten zu speichern, z.B. Name und Telefonnummer eines Dozenten ("Attribute"). Der Name ist Schlüssel (Markierung "#").

- Dozenten halten Vorlesungen.

  Beziehung / "Relationship" zwischen Dozenten und Vorlesungen.

Beispiel (2)

Möglicher Zustand:



Es gibt in diesem Zustand ein Entity (Objekt) vom Typ "Dozent", dies hat z.B. den Wert "Brass" für das Attribut "Name". Entsprechend gibt es zwei Entities vom Typ "Vorlesung". Das vorhandene Dozent-Entity steht zu beiden Vorlesungs-Entities in der Beziehung "hält".

Beziehung zur Logik

- Entity-Typen ("Klassen") werden im Zustand interpretiert als endliche Mengen (von "Entities").

  Entity-Typen sind aus Sicht der Logik also Sorten (Typ-Namen).
- Attribute werden interpretiert als Abbildungen von einem Entity-Typ auf einen Datentyp.

  Die Abbildung ordnet jedem Entity einen Attributwert zu. Attribute sind also Namen für Funktionen (von Entity-Sorte in Datentyp).
- Relationships werden interpretiert als Relation zwischen den beiden beteiligten Entity-Typen.

  D.h. als Menge von Paaren aus jeweils einem Entity der beiden Typen. Ein Relationship ist also ein Prädikat der Stelligkeit 2.

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 (Entity) kann nicht gespeichert werden, aber ihr Name (Datentyp-Element).
- Die meisten DBMS haben eine vordefinierte Menge an Datentypen. Im ER-Entwurf kann man aber auch Nicht-Standard-Datentypen verwenden.

  Der ER-Entwurf soll ja nicht an ein spezielles DBMS gebunden sein.

Begriffe des ER-Modells (3)

Attribute:

- Eine Eigenschaft oder Charakteristik eines Entities.
- 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.
- Attribute können optional sein, d.h. Nullwerte erlauben.

  Sie werden dann als partielle Funktion interpretiert. In der Barker-Notation werden optionale Attribute mit “◦” vor dem Namen markiert.

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”.

Entity-Typen: Notation (1)

- In der Barker-Notation wird für Entity-Typen eine “Softbox” verwendet, d.h. ein Rechteck mit abgerundeten Kanten.
- In das Rechteck wird oben zentriert der Name des Entity-Typs geschrieben:

Es ist üblich, als Entity-Typ-Name die Singular-Form eines Nomens zu benutzen. Im Buch von Barker und im Werkzeug “Oracle Designer” werden Entity-Typen immer ganz in Großbuchstaben geschrieben.

Attribute: Notation (1)

- Attribute werden in der Softbox des Entity-Typs aufgelistet, zu dem sie gehören.

  In Entwurfswerkzeugen kann man üblicherweise wählen, ob Attribute angezeigt werden sollen. Insofern ist auch das Diagramm auf der letzten Folie ohne Attribute korrekt, und bedeutet nicht automatisch, dass der Entity-Typ keine Attribute hat.

- Man schreibt sie unterhalb des Entity-Typ-Namens (eine Zeile pro Attribut):

Attribute: Notation (2)

- Jedem Attribut wird genau eins der folgenden drei Symbole vorangestellt:

  #: Dieses Attribut ist Teil des Primärschlüssels.

  Der Primärschlüssel des Entity-Typs ist die Kombination aller mit # markierten Attribute. Man kann im Diagramm also nur einen Schlüssel pro Entity-Typ darstellen.

  ∗: Dieses Attribut erlaubt keine Nullwerte.

  Primärschlüssel erlauben natürlich auch keine Nullwerte. In der Original-Notation im Buch von Barker wurde "# ∗" zusammen angegeben. Beim Werkzeug "Oracle Designer" wurde dies vereinfacht.

  ○: Das Attribut ist optional (erlaubt Nullwerte).

Relationships: Notation (1)

- Ein Relationship zwischen zwei Entity-Typen wird durch eine Verbindungslinie zwischen den Entity-Typen notiert:

- Strichelung (hier links) und Aufspaltung (Krähenfuß, hier rechts) können zunächst ignoriert werden.

  Sie drücken Kardinalitäten aus, eine spezielle Art von Integritätsbedingungen für Relationships. Diese werden ab Folie 8-62 erläutert.

Relationships: Notation (2)

- Ein Relationship hat immer zwei Namen, einen für jede Richtung.

  Man schreibt immer den Namen für die Beziehung von A nach B in die Nähe von A, und den Namen für die Beziehung von B nach A in die Nähe von B. Bei horizontalen Verbindungen ist es üblich, den Beziehungsnamen links über die Verbindungslinie zu schreiben, und rechts unter die Verbidnungslinie. Das ist aber keine Vorschrift.

- Rekursive Relationships (Beziehungen von einem Entity-Typ zum gleichen Entity-Typ) sind zulässig.
  - ◇ Beispiel: Die "ist Voraussetzung für"-Beziehung zwischen zwei Vorlesungen.

Eindeutigs-Bedingungen

- Die Namen von Entity-Typen im Diagramm müssen eindeutig sein (d.h. jeder Kasten muss einen verschiedenen Namen enthalten).
- Ein Entity-Typ darf keine zwei Attribute mit gleichem Namen haben.

  Verschiedene Entity-Typen können gleich benannte Attribute haben.
- Relationships werden über die beiden beteiligten Entity-Typen und die beiden Namen identifiziert.

  Man kann also Relationship-Namen mehrfach verwenden. Es darf nur nicht zwischen den gleichen beiden Entity-Typen zwei Relationships geben, die in beiden Namen übereinstimmen.

Diagramme und Schemata (1)

- In den ER-Diagrammen werden nicht alle Schema-Informationen dargestellt:
  - ◇ Attribute haben Datentypen, die in vollständigen ER-Schemata festgelegt werden müssen.
  - ◇ Nur ein Schlüssel pro Entity-Typ darstellbar.
  - ◇ Große ER-Diagramme werden übersichtlicher, 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.

Diagramme und 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 $\mathcal{I}[A]: \mathcal{I}[E] \rightarrow val(D)$ für jedes Attribut A eines Entity-Typs E, wobei D der Datentyp von A und $val(D)$ die Domain (Wertemenge) von D ist,

  Wenn das Attribut Nullwerte erlaubt (Markierung mit “∘”), handelt es sich um eine partielle Abbildung. Alternativ kann man eine totale Abbildung in $val(D) \cup \{\text{null}\}$ verwenden.
- 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,

ER-Schemata: Semantik (2)

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

  Die logische Interpretation enthält formal auch die Interpretation der Datentypen, diese ist aber fest (gehört nicht zum Zustand).

- 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

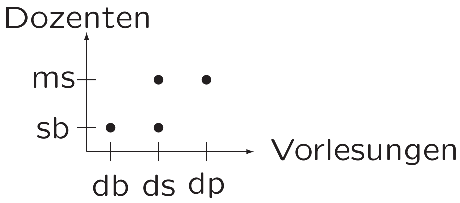
  $g_2(db) =$ 'Datenbankverwaltung',

  $g_2(ds) =$ '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:



Schlüssel (1)

- Selbstverständlich muss ein gültiger DB-Zustand auch alle Integritätsbedingungen erfüllen, die im Schema spezifiziert sind.
- 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 die Matrikelnummer ein Schlüssel für die Studenten einer Universität: Zwei verschiedene Studenten haben nie die gleiche Matrikelnummer.

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:

- Ein Entity-Typ kann mehrere Schlüssel haben, aber im Diagramm kann man nur einen angeben, den sogenannten Primärschlüssel.
- Die Attribute, die den Primärschlüssel bilden, werden mit “#” markiert:

```
DOZENT
# Vorname
# Nachname
* Tel
```


Schlüssel (4)

Spezifikation mit Logik:

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

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

- Äquivalente Formulierung:

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

Schlüssel (5)

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 (6)

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.
- In der Literatur üblich: Schlüssel-Definition enthält die Minimalität der Menge von Schlüsselattributen (zusätzlich zur Eindeutigkeitsbedingung).

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

Schlüssel (7)

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.

Schlüssel (8)

Auswahl des Primärschlüssels:

- Wenn es mehrere Schlüssel gibt, sollte man als Primärschlüssel möglichst einen wählen,
  - ◇ der nur aus einem Attribut besteht, und
  - ◇ möglichst 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 besonders schnell sein. Ansonsten ist die Wahl nicht sehr wichtig.
- Die übrigen Schlüssel werden Alternativ-, Sekundär- oder `UNIQUE`-Schlüssel genannt.

  Da sie nicht im Diagramm angegeben werden können, müssen sie in zusätzlicher Dokumentation spezifiziert werden.

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)

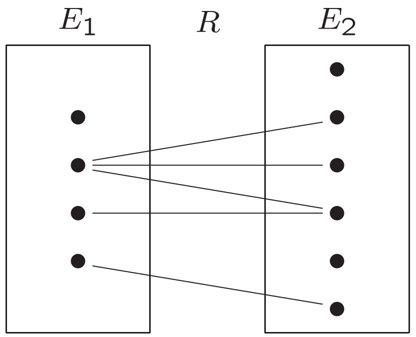
- Der Zweck von Schlüsseln ist nicht nur die Identifikation von Entities.
- Schlüssel sollten auch helfen, Duplikate in der DB zu vermeiden. Künstliche Schlüssel tun dies nicht.
- Oft sind Duplikate keine exakten Kopien: Z.B. andere Abkürzungen oder Großschreibung verwendet.

  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.

Inhalt

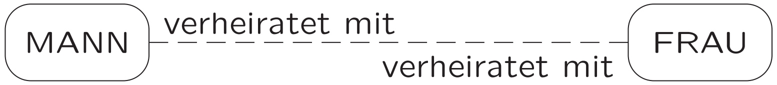
Kardinalitäten (1)

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)

Klassifikation über Maximalzahl verbundener Entities:

- Eins-zu-eins-Beziehungen ("1:1"):
  In beiden Richtungen ist ein Entity mit höchstens einem Entity des anderen Typs verbunden.
- Eins-zu-viele-Beziehungen ("1:n"):
  In einer Richtung kann es mehrere Entities geben, in der anderen Richtung nur eins.
- Viele-zu-viele-Beziehungen ("n:m"):
  In beiden Richtungen kann ein Entity mit mehreren Entities des anderen Typs in Beziehung stehen.

Kardinalitäten (4)

Beispiele/Notation:

- In der Barker-Notation wird die Klassifizierung der Relationships in "1:1", "1:n" und "n:m" über sog. "Krähenfüße" an den Enden ausgedrückt:

- Dies ist eine "Eins-zu-viele Beziehung":
  - ◇ Ein Dozent kann viele Vorlesungen halten.
  - ◇ Jede Vorlesung wird nur von einem Dozenten gehalten (weil auf Dozentenseite kein Krähenfuß).