Part 3: Introduction to the Entity-Relationship Model

References:

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

Objectives

After completing this chapter, you should be able to:

- explain the three phases of database design.

  Why multiple phases are useful?

- evaluate the significance of the ER-model for DB design.
- enumerate the basic constructs of the ER-model.
- develop ER-diagrams (schemas in the ER-model) for a given (small) application.
- compare and evaluate given ER-diagrams.

Overview

1. Database Design Overview
2. Basic ER-Constructs
3. Integrity Constraints: General Remarks
4. Kinds of Relationships (Cardinalities)
5. Keys, Weak Entities
6. Quality of ER-Schemas

Database Design (1)

- Overall goal: Develop programs to support given real world tasks.
- These programs need persistently stored data.
- Methods from software engineering should be used (specialized for such data-intensive programs).
- Database design is the process of developing a database schema for a given application.

  It is a subtask of the overall software engineering effort.

Database Design (2)

- The specification of programs and data is intertwined, both depend on each other:
  - ◇ The schema should contain the data needed by the programs.
  - ◇ Programs are often easy to specify once one has specified the data to be manipulated by them.
- Data is an independent resource:
  - ◇ Often later additional programs will be developed based on the collected data.
  - ◇ Also, ad-hoc queries may be used.

Database Design (3)

- During DB design, a formal model of some aspects of the real world ("Miniworld", "Domain of Discourse") must be built.

  Questions about the real world should be answered from the database.
  A list of such questions can be an important input for database design.

- The real world is the measure of correctness for the schema: DB states should correspond to states of the real world.

Database Design (4)

- Database design is not easy:
  - ◇ The designer must learn about the application domain.
  - ◇ Exceptions: The real world is very flexible.
  - ◇ Size: Database schemas can be very big.
- As any complicated task, DB design is done in several steps.

Database Design (5)

- There are usually three schema design phases:
  - ◇ Conceptual Database Design produces the initial model of the miniworld in a conceptual data model (like the Entity-Relationship-Model).
  - ◇ Logical Database Design transforms this schema into the data model supported by the DBMS to be used (typically the relational model).
  - ◇ Physical Database Design aims at improving the performance of the final system.

    Indexes and storage parameters are selected during this phase.

Database Design (6)

Why multiple design phases?

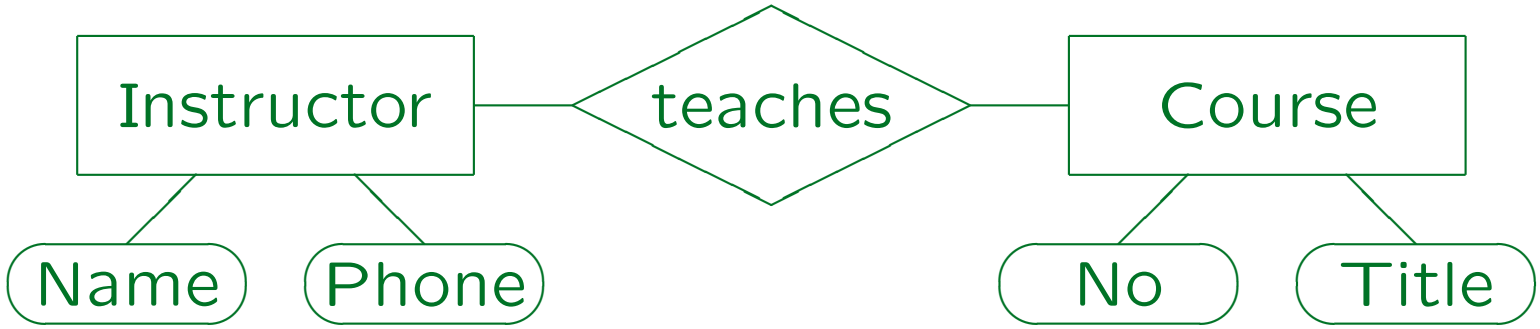
- Allows problems to be separated and attacked one after the other.
- E.g., during conceptual design, there is no need to worry about performance aspects or limitations of a specific DBMS.

  Focus is on producing a correct model of the real world.
- DBMS features do not influence conceptual design, and only partially influence the logical design.

  Thus, the conceptual design is not invalidated, if a different DBMS is later used.

Example (1)

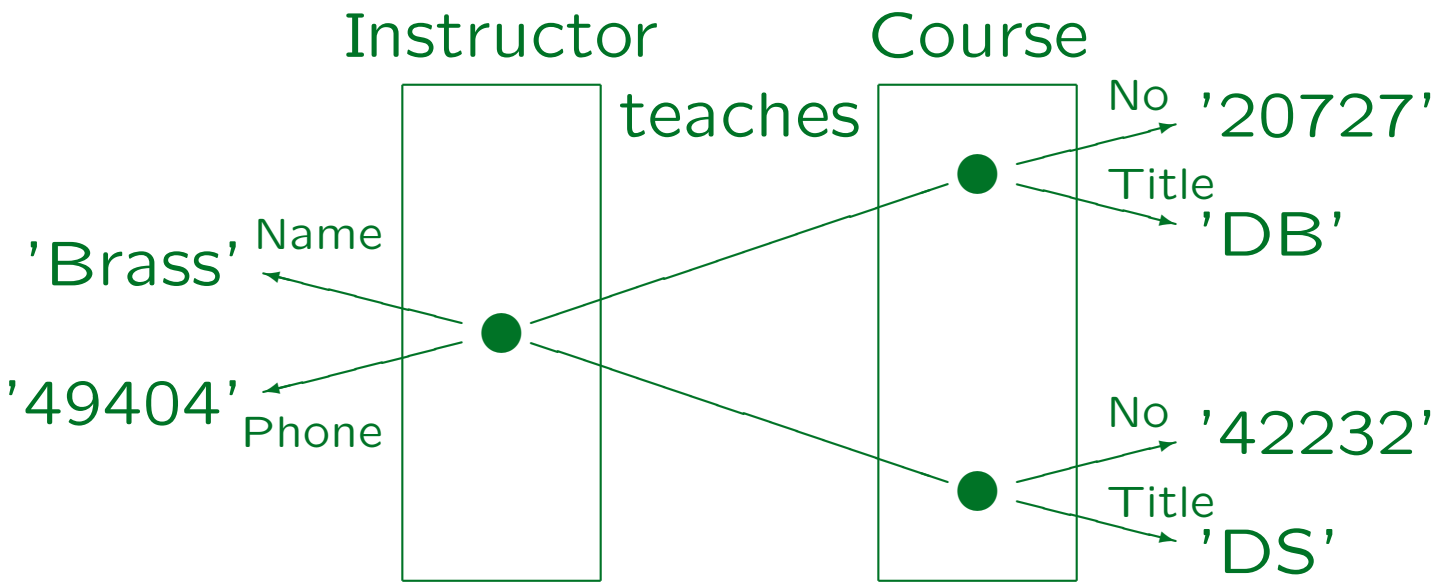
ER-Schema in Graphical Notation:



- This miniworld contains instructors and courses.
- Instructors teach courses.
- Instructors have a name and a phone number.
- Courses have a number (like "20727") and a title.

Example (2)

Possible State:



The ER-Model (1)

- The Entity-Relationship-Model is called a "semantic data model", because it more closely resembles the real world than e.g. the relational model.
  - ◇ In the ER-model, persons are modelled. In the relational model, only their names/numbers.
  - ◇ In the ER-model, there is a distinction between entities and relationships. In the relational model, both are represented by relations/tables.

    This expressiveness is not needed to satisfy the information requirements of the applications. But it makes the correspondence between the schema and the real world clearer (like a comment).

The ER-Model (2)

- Proposed by Peter Pin-Shan Chen (1976).
- There is a useful graphical notation which helps to establish a better overview; to "see" the structure of the data.

  It also helps to communicate with the future users.
- There is no commercial entity-relationship DBMS.

  A schema transformation into another data model is unavoidable. However, object-oriented DBMS are quite similar.
- Many variations and extensions of the ER-Model have been proposed.

The ER-Model (3)

- There are specialized graphical editors and other design tools.

  E.g. Oracle Designer is a CASE tool for DB-applications, and one component is the ER Diagrammer. (CASE: Computer-Aided Software Engineering.) Alternatives: Sybase PowerDesigner, CA ERwin, ...

- The ER-model is a standard tool for conceptual DB design. However, recently, object-oriented methods are also used.

  Many people believe that UML (unified modelling language) is "the future". All design formalisms are based on the ER-model. Knowledge of the ER-model is an important foundation for any DB design.

Overview

Basic ER-Model Concepts (1)

Entities:

- Objects in the miniworld about which information has to be stored. E.g. persons, books, courses.

  It does not matter whether entities have a physical existence (and can be touched) or only a conceptual existence.

- At each instant, the miniworld that has to be modelled can contain only a finite number of entities.

  E.g. "all numbers" (infinitely many) cannot be entities.

- It must be possible to distinguish entities from each other, i.e. they must have some identity.

  So ants in a heap do not qualify as entities.

Basic ER-Model Concepts (2)

Data Type Elements:

- Values from some possibly infinite set, which can be stored and printed.
- E.g. strings, numbers, dates, lengths, pictures.
- A person cannot be stored (an entity), but his/her name can be stored (a data type element).
- Most current DBMS have some predefined set of data types which they support.
- It is possible to use non-standard types in ER-schemas (complicates the later logical design).

Basic ER-Model Concepts (3)

Attribute:

- A property or characteristic of an entity.

  Also relationships can have attributes, see below.

- E.g. the title of this course is “Databases I”.
- The value of an attribute is an element of a data type like string, integer, date: It has a printable representation.

Basic ER-Model Concepts (4)

Relationship:

- Relation between pairs of entities.

  Such relationships are also called "binary relationships" in order to distinguish them from relationships in which more than two entities participate. Many authors (but few tools) permit relationships of arbitrary arity (e.g., ternary relationships). My experience shows that this often leads to errors (students "merge" two binary relationships into one ternary, this violates normal forms).

- E.g. I (a person) teach "Databases I" (a course).
- The word "Relationship" is also used as an abbreviation for "Relationship-Type" (see below).

  It should be clear from the context what is meant.

Basic ER-Model Concepts (5)

Entity-Type:

- Set of similar entities (with respect to the information that has to be stored about them), i.e. entities which have the same attributes.
- E.g. all faculty members of this university.

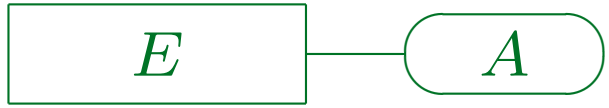
Relationship-Type:

- Set of similar relationships.
- E.g. “X teaches course Y”.

ER-Diagrams (1)

- Entity-Type E:

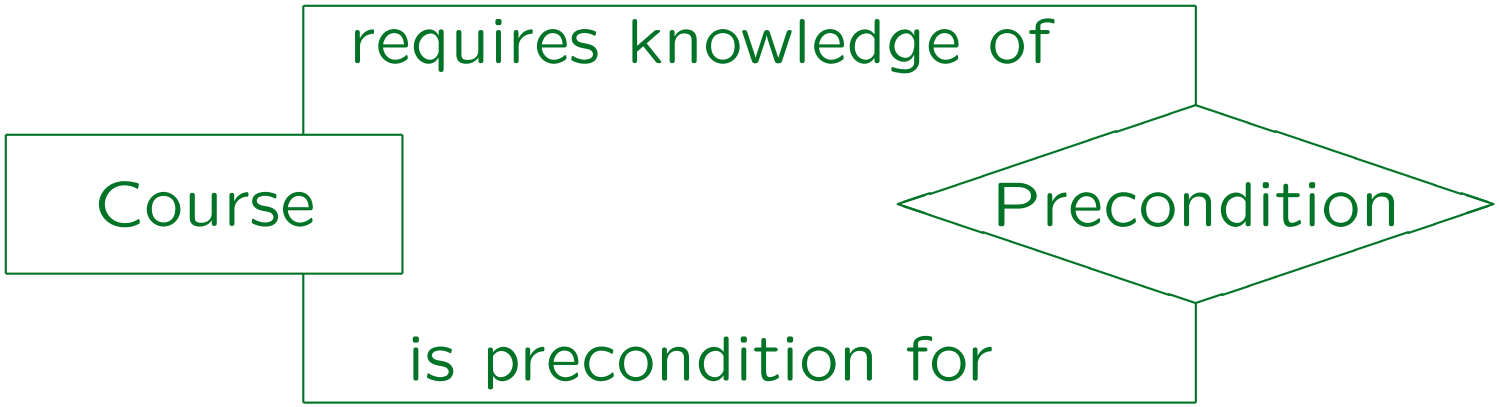
- Attribute A of Entity-Type E:



- Relationship R between Entity-Types E_1 and E_2:

ER-Diagrams (2)

- Relationships can also exist between entities of the same type ("recursive relationships"):

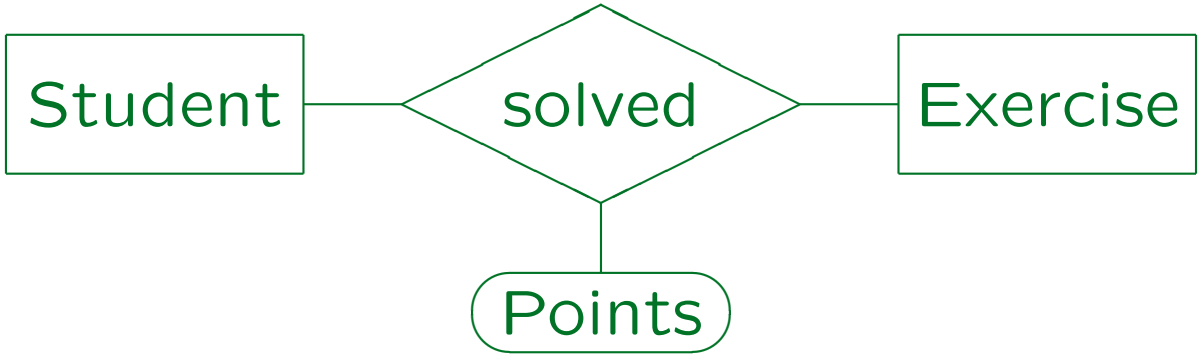


- In this case, "role names" must be attached to the connecting edges.

  Roles may be indicated in this way for any relationship.

ER-Diagrams (3)

- Relationships can have attributes, too:



- This means that a number of points is stored for every pair of student X and exercise Y such that X submitted a solution to Y.

Graphical Syntax (1)

- An ER-Diagram contains
  - ◇ boxes,
  - ◇ diamonds,
  - ◇ ovals,

  plus interconnecting lines.
- Boxes, diamonds, and ovals are each labelled by a string.

  The string is written into the construct.

Graphical Syntax (2)

- Interconnecting lines are only allowed between
  - ◇ a box and a diamond,
  - ◇ a box and an oval, and
  - ◇ a diamond and a oval.
- In addition, these constraints must be satisfied:
  - ◇ A diamond must have exactly two connecting lines to boxes. There may be any number to ovals.
  - ◇ An oval must have exactly one connecting line.

Graphical Syntax (3)

- The names of boxes (entity types) must be unique in the entire diagram.

  I.e. there cannot be two boxes with the same label.

- The names of ovals (attributes) must only be unique for a single box or diamond.

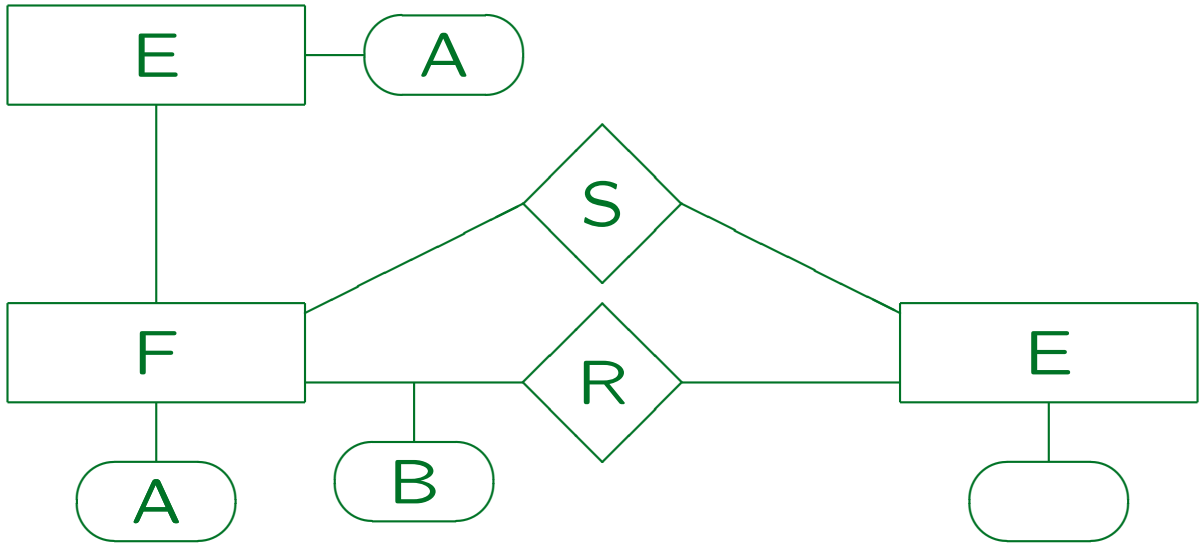
  I.e. there cannot be two ovals with the same name which are linked to the same box (or the same diamond).

- Diamonds (relationships) must be uniquely identified by name and their connections to boxes.

  I.e. there cannot be two diamonds which are linked to the same two boxes and have the same name.

Exercise

Which errors does this diagram contain?



Exercise

Define an ER-schema (diagram) for the following application:

- Information about researchers in the database field must be stored.
- For each researcher, his/her last name, first name, email address, and homepage (URL) is needed.
- Also his/her current affiliation (employer) is needed (assume that all researchers work at universities).
- For each university, its name, URL, and country should be stored.

ER-Schemas (1)

- Usually, not all schema information is denoted in an ER-diagram:
  - ◇ The attributes have a data type, which must be specified in a complete ER-schema.
  - ◇ Sometimes an ER-diagram becomes clearer if the attributes are not shown.

    It is not uncommon that an entity-type has 50 attributes.
  - ◇ Comments/Explanations are useful, but do not look good in ER-diagrams.

ER-Schemas (2)

- There should exist a "real schema" (e.g. in textual form or in a database). The ER-diagrams are then only excerpts used for illustration purposes.

  However, there is no agreement for a textual syntax for writing down complete schemas, whereas there is some agreement on ER-diagrams.

- The important thing to learn is the graphical syntax (and the fact that it might not show everything).
- Modern database design tools solve the problem: E.g. clicking on an entity-type in the diagram shows further information in dialog boxes.

ER-Schemas: Semantics (1)

A DB state $\mathcal{I}$ interprets the symbols in the schema by defining

- a finite set $\mathcal{I}[E]$ for every entity-type E,
- a mapping $\mathcal{I}[A]: \mathcal{I}[E] \rightarrow val(D)$ for every attribute A of an entity-type E, where D is the data-type of A and $val(D)$ is the domain (value set) of D,
- a relation $\mathcal{I}[R] \subseteq \mathcal{I}[E_1] \times \mathcal{I}[E_2]$ for every relationship R between entity types E_1 and E_2,
- a mapping $\mathcal{I}[A]: \mathcal{I}[R] \rightarrow val(D)$ for every attribute A of a relationship R (D is the data type of A).

ER-Schemas: Semantics (2)

- As can be expected, this is a special case of a logical interpretation as introduced in Chapter 2.

  Only relationship attributes are difficult to handle in standard logic.
  The interpretation of the data types was separated from the DB state.

- The main restrictions of the ER-Model are:
  - ◇ Besides the given data types, only new sorts with finite domains can be introduced (entity types).
  - ◇ The new functions must be from entity types (or relationships) to data types (attributes).
  - ◇ The new predicates must have entity types as arguments and must have arity 2 (relationships).

ER-Schemas: Semantics (3)

Example State:

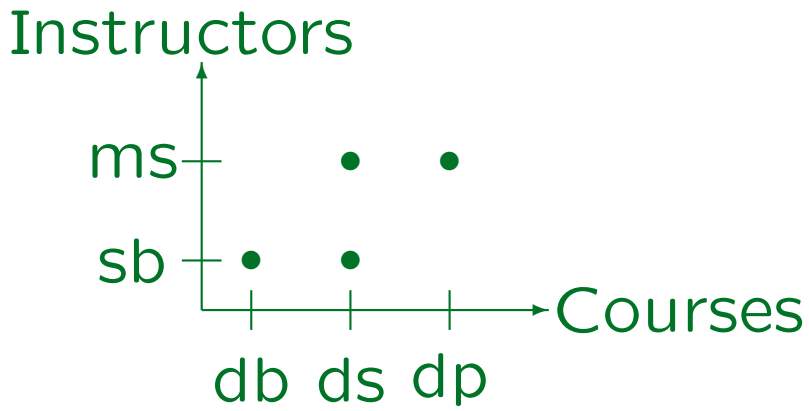
- $\mathcal{I}[Instructor] = \{sb, ms\}$
- $\mathcal{I}[Name] = f_1$, where
  $f_1(sb) =$ 'Brass', $f_1(ms) =$ 'Spring'.
- $\mathcal{I}[Phone] = f_2$, where
  $f_2(sb) = 49404$, $f_2(ms) = 49429$.

ER-Schemas: Semantics (4)

- $\mathcal{I}[Course] = \{db, ds, dp\}$
- $\mathcal{I}[No] = g_1$, where
  $g_1(db) = 20727$, $g_1(ds) = 42232$, $g_1(dp) = 40492$.
- $\mathcal{I}[Title] = g_2$, where
  $g_2(db) =$ 'Database Management',
  $g_2(ds) =$ 'Data Structures',
  $g_2(dp) =$ 'Document Processing'.

ER-Schemas: Semantics (5)

- $\mathcal{I}[teaches] = \{(sb, db),\ (sb, ds),\ (ms, ds),\ (ms, dp)\}$.
- The cartesian product $\times$ constructs the set of all pairs, e.g. $\mathbb{R} \times \mathbb{R}$ is the set of all (X, Y)-pairs where X and Y are real numbers.
- Here: (X, Y)-pairs with instructor X and course Y:



ER-Schemas: Semantics (6)

Consequences for Attributes:

- There are extensions, but the basic notion of an attribute requires that
  - ◇ Attribute values are defined.

    No unknown values.
  - ◇ Attributes are single valued.

    No multiple/set values.
  - ◇ Attribute values are atomic.

    No record structures (unless data types have already this structure, e.g. date values). But the basic ER-model treats all attribute values as atomic, i.e. it adds no record structures besides those already given by the data types.

ER-Schemas: Semantics (7)

Consequence for Relationships:

- A relationship is interpreted by a set of entity-pairs.
- A set either contains an element or does not contain it. It cannot be contained "two times".
- Therefore, a relationship either exists between two entities or it does not exist between these entities.

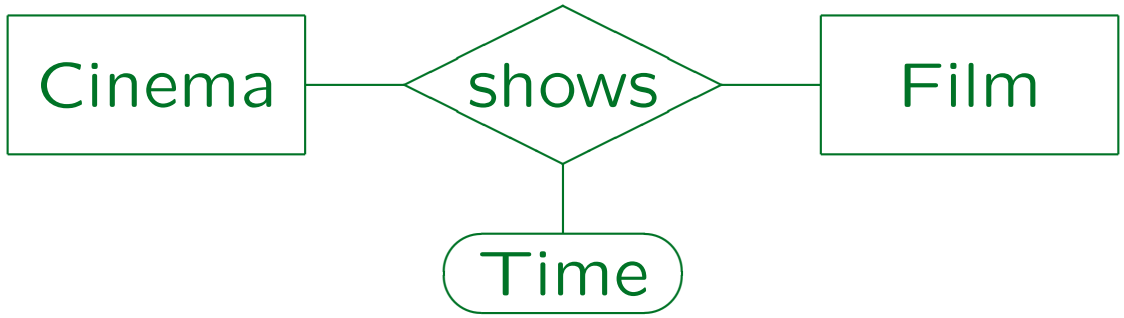
  There cannot be multiple connections between the same two entities with respect to the same relationship type.

- So if the relationship has an attribute, its value must be unique for any pair of entities.

ER-Schemas: Semantics (8)

Example/Exercise:

- Consider this schema:

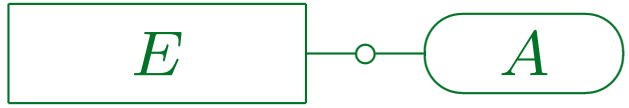


- Suppose a cinema shows the same film at 3pm and at 6pm.
- Can this information be stored in the given schema?

  ☐ Yes ☐ No

Optional Attributes

- One can also permit attributes that are only partially defined (optional attributes, can be null).
- Such attributes can be marked by a small circle on the line between the entity type (or relationship) and the attribute:

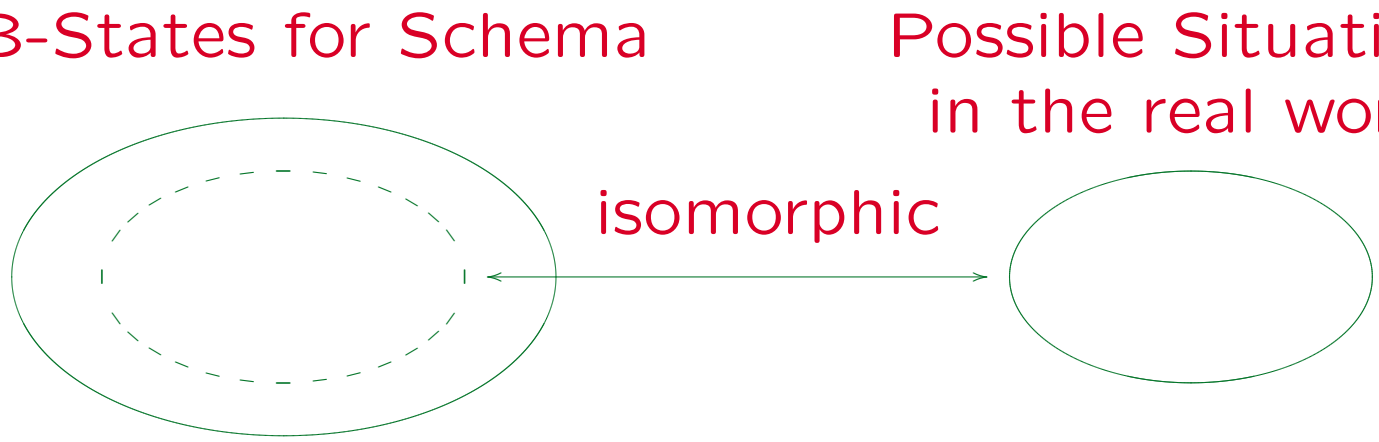


- The semantics of this attribute in a state I is a partial function from $I(E)$ (set of all E-entities) to $val(D)$ where D is the data type of A.

Overview

Valid Database States (1)

- Goal of DB-Design: The database should be an image of the relevant subset of the real world.
- But the definition of the basic model structure (entities, attributes, and relationships) allows often too many (meaningless, illegal) database states:



Valid Database States (2)

CUSTOMER				
Cust_No	Name	Birth_Year	City	···
1	Smith	1936	Pittsburgh	···
2	Jones	1965	Philadelphia	···
3	Brown	64	New York	···
3	Ford	1990	Washington	···

- Customer numbers must be unique.
- The year of birth must be greater than 1870.
- Customers must be at least 18 years old.

Valid Database States (3)

- Two kinds of errors must be distinguished:
  - ◇ Entering wrong data, i.e. the DB state corresponds to a different situation of the real world than the actual one.

    E.g., 8 points given for Homework 1 in the DB vs. 10 in the real world. Then the DB state is wrong, but not the schema.
    Exercise: What can be done to guard against such errors?
  - ◇ Entering data which do not make sense, or are illegal.

    E.g. -5 points for some homework, or two entries for the same student and same exercise. If such impossible data can be entered, the DB schema is wrong (design error).

Valid Database States (4)

- If the DB contains illegal/meaningless data, it becomes inconsistent with our general understanding of the real world.
- If a programmer assumes that the data fulfills some condition, but it actually does not, this can have all kinds of strange effects (including the loss of data).

  E.g. the programmer assumes that a certain column cannot contain null values (a special value that indicates empty table entries, see Chapter 4). So he/she uses no indicator variable when fetching data. As long as there are no null values, this works. But if the schema does not prevents this, after some time, somebody will enter a null value. Then the program will crash (with a user-unfriedly error message).

Constraints (1)

- Integrity Constraints (or short "Constraints") are conditions which every database state must satisfy.
- This restricts the set of possible database states.

  Ideally to images of possible real world situations.
- Integrity constraints can be specified as part of the database schema.
- The DBMS will refuse any change which would violate such a constraint.

  In this way, invalid states are excluded. Current DBMS do not yet permit arbitrary constraints, only some special cases, see below.

Constraints (2)

- Each data model has special support/notation for certain common kinds of constraints.

  For instance, in the Entity-Relationship-Model, keys, cardinality constraints, and the restrictions related to weak entities have a special graphical syntax. These types of constraints will be explained below.

- Even if constraints are needed that are not specially supported by the data model or the DBMS, they should be documented as part of the DB design.

  E.g. as a logical formula or simply in natural language. One can also write a query that returns the violations of the constraint (i.e. the condition is then that the result of this query is always empty). Future systems will probably directly support more general constraints.

Constraints (3)

- Enforcement of general constraints:
  - ◇ The application programs used for entering the data check the constraint.
  - ◇ Changes are only done via stored procedures in the DBMS, which check the constraint.
  - ◇ The DBMS automatically executes special procedures attached to a table (triggers) whenever the table data is changed. These do the check.
  - ◇ From time to time, a query is executed that finds constraint violations.

Constraints (4)

Exercise:

Consider an instructor/professor-course database:

Which of the following are constraints?

- It is possible that an instructor teaches no course.
  - ☐ Yes ☐ No
- An instructor can teach at most four courses.
  - ☐ Yes ☐ No

Constraints (5)

Exercise, continued:

- The attribute "Type" of "Instructor" can only have the values 1, 2, 3, 4, 5.
  ☐ Yes ☐ No
- The "Type" value means the following: 1: Teaching Fellow, 2: Adjunct Faculty, 3: Assistant Professor, 4: Associate Professor, 5: Full Professor.
  ☐ Yes ☐ No
- Course titles should be unique.
  ☐ Yes ☐ No

Trivial/Implied Constraints

- A trivial constraint is a condition which is always satisfied (logically equivalent to "true").
- A constraint A logically implies a constraint B if whenever A is true, also B is true.

  I.e. the states satisfying A are a subset of the states satisfying B. This is the standard definition of logical implication.

- E.g. A: "Every instructor teaches 1 or 2 courses".
  B: "Instructors can teach at most 4 courses".
- Trivial/implied constraints should not be specified.

  Adds complication, but does not change the set of valid states.

Relation to "Business Rules"

- "Business rules" are similar to constraints: They are rules that must be followed by employees (to prevent chaos).
- Constraints are used to implement "business rules".

  Certain rules, e.g. that customers must be 18 years old or that advanced cryptology software is not sold to customers outside the US, can be enforced by not allowing entry of such an order into the database. A state violating the business rules is considered invalid.

- However, there are also "business rules" that are implemented via the basic model structure, access rights, application programs, etc.

Summary (1)

Why specify constraints?

- Some protection against data input errors.
- Constraints document knowledge about DB states.
- Enforcement of laws / company standards.
- Protection against inconsistency if redundant data is stored (i.e. the same information is stored twice).
- Queries/programs become simpler if the programmer is not required to handle the most general cases (i.e., cases where the constraint is not satisfied).

  E.g., if columns are known to be not null: no indicator variable.

Summary (2)

Constraints and Exceptions:

- Constraints cannot have any exceptions.

  Any attempt to enter data that violates a constraint will be rejected.

- One can expect that eventually there will be exceptional situations in which the DBS seems unflexible because of the specified constraints.

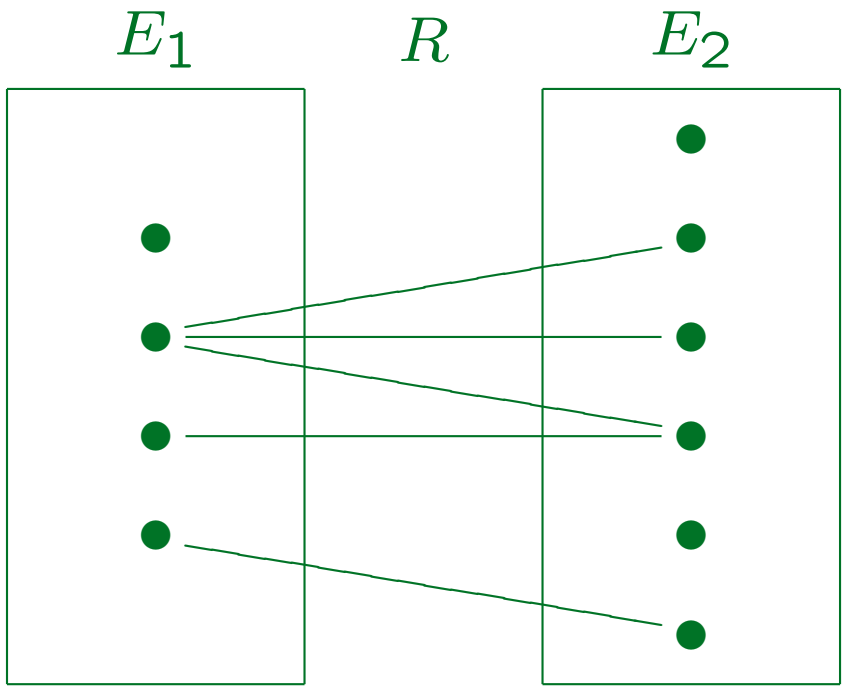
- Only conditions that are unquestionable should be defined as constraints.

  "Normally true" conditions might be useful, but not as constraints. E.g. programs can ask for an additional confirmation if a new customer is over 100 years old. Also useful information for physical design.

Overview

Cardinalities (1)

General Relationship:



Cardinalities (2)

- In general, there is no restriction on how often an entity participates in a relationship.
- An entity can be connected to one entity of the other type, to more than one, or it can have no partner at all.
- However, often we have some knowledge of how many E_2-entities an E_1-entity can be related.

Cardinalities (3)

- In the (min,max)-Notation, one specifies an interval for the number of outgoing edges of every entity:

- I.e. an entity of type E_1 may be related to between m_1 and n_1 entities of type E_2.
- m_2 is the minimal number of E_1 entities, to which an E_2 entity must be related, and n_2 is the maximal number to which it may be related.

Cardinalities (4)

Formal Semantics:

- For every DB state I and every entity $e_1 \in I(E_1)$:

$$m_1 \leq |\{e_2 \in I(E_2) \mid (e_1, e_2) \in I(R)\}| \leq n_1.$$

- For every DB state I and every entity $e_2 \in I(E_2)$:

$$m_2 \leq |\{e_1 \in I(E_1) \mid (e_1, e_2) \in I(R)\}| \leq n_2.$$

Extension:

- "$*$" can be used as maximum if there is no limit.
- $(0, *)$ means no restriction at all (min: 0, max: ∞).

This cardinality is the default (if no cardinality is specified).

Cardinalities (5)

Example:

- A man can be married to at most one woman and vice versa:

- An airport lies in exactly one country, a country can have many airports (it might be possible that there are countries without airport):

Cardinalities (6)

Exercise: Define Cardinalities.

- Besides normal customers, the database contains also "customers" who have not yet ordered anything, but only got the product catalog.

- An order can be about several products:

Selecting Cardinalities (1)

- Draw an example of a valid database state: Some entities of type E_1, some of type E_2, and edges between related entities.
- Now count the outgoing edges for all entities of type E_1. (Later do the same with E_2.)
- E.g. in the relationship depicted on slide 3-55, there are E_1-entities with 0, 1, and 3 outgoing edges.
- This means that the strongest possible cardinality restriction is $(0, 3)$. This is satisfied by the example state.

Selecting Cardinalities (2)

- Knowledge of the application might lead to weaker restrictions (i.e. larger intervals), e.g. $(0, 5)$ or even $(0, *)$.

  (a, b) is weaker than (c, d) if $a \leq c$ and $d \leq b$.

- In general, the cardinalities $(0, 1)$, $(1, 1)$, and $(0, *)$ are especially common and easy to enforce in a relational schema.

  E.g. instead of $(0, 30)$, it might be easier to choose $(0, *)$. However, if 30 is a hard limit (i.e. the database state would be invalid if it is violated), one should specify the cardinality $(0, 30)$ and enforce it via constraints, triggers, or checks in application programs.

Common Cases (1)

- Normally, the minimum cardinality will be 0 or 1, and the maximum cardinality will be 1 or $*$.

  So only $(0,1)$, $(1,1)$, $(0,*)$, $(1,*)$ are common in practice. Of these, $(1,*)$ is difficult to enforce in a relational schema.

- In order to understand a relationship, one must know the cardinality specifications on both sides.
- The maximum cardinalities are used to distinguish between "many to many", "one to many" / "many to one", and "one to one" relationships.

Common Cases (2)

"Many to Many" Relationships:

- Both maximum cardinalities are "$*$":

  The minimum cardinalities can be 0 or 1, see participation below.

- One student can be enrolled in many courses, and one course can have many students.
- This is the most general case.
- When translated into the relational model, "many to many" relationships will need an extra table.

Common Cases (3)

"One to Many" Relationships:

- Maximum cardinality 1 on the "many" side and $*$ on the "one" side.

- E.g. one instructor teaches many courses, but each course has at most one instructor ("one instructor, many courses").

  So this is "one to many" from instructor to course.

Common Cases (4)

"One to Many" Relationships, Continued:

- Note that 1/one and $*$/many are inversed!
  - "Instructor" is the "one" side, but has maximum cardinality $*$.
  - "Course" is the "many" side, but has maximum cardinality 1.
- "One to many" relationships do not need an extra table when translated into the relational model.

  They are probably the most common kind of relationship.
- "Many to one": symmetric (e.g. "taught by").

Common Cases (5)

"One to One" Relationships:

- Maximum cardinality 1 on both sides:

- A man can be married to at most one woman, a woman can be married to at most one man.

  Or: "Is head of" between employee and department: Every department has exactly one head (total participation), every employee can lead at most one department (normal employees lead no department: partial participation).

Common Cases (6)

Participation:

- The minimum cardinalities define whether the participation of entities in the relationship is
  - ◇ total (mandatory): Every entity must participate in the relationship.
  - ◇ partial (optional): Some entities participate in the relationship, others do not.
- Minimum cardinalities are not important for the classification of a relationship as "many to many", "one to many", or "one to one".

Common Cases (7)

- In the following example,
  - ◇ the participation of "Course" is mandatory (every course must have an instructor),

    This might be too restrictive in practice. It will hold when the term starts, but might not hold during planning.
  - ◇ the participation of instructor is optional (instructors can be on sabbatical).

    I.e. there can be faculty members that do not teach courses in the current term.

Specification in Logic

- Maximum cardinality 1: Every course is taught by at most one instructor:

  $$\forall \texttt{course C}, \texttt{instructor I1}, \texttt{instructor I2}: \texttt{teaches(I1, C)} \wedge \texttt{teaches(I2, C)} \rightarrow \texttt{I1} = \texttt{I2}.$$

- Maximum cardinality $*$: This is no constraint.
- Minimum cardinality 1: Every course is taught by at least one instructor:

  $$\forall \texttt{course C}: \exists \texttt{instructor I}: \texttt{teaches(I, C)}$$

- Minimum cardinality 0: This is no constraint.

Alternative Notations (1)

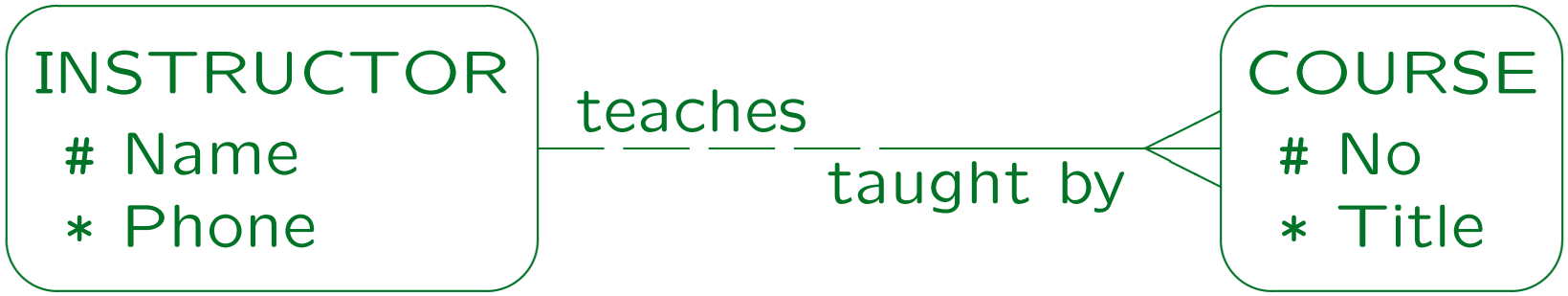
- There are many variants of the ER-Notation. Cardinalities are one point in which they differ quite significantly.
- E.g. one can use only "many to many" (N:M), "one to many" (1:N), and "one to one" (1:1).

- Participation is often not specified.

  Sometimes a double line is used to indicate mandatory participation.

Alternative Notations (2)

- Oracle Designer uses the "crowsfoot" notation:



The "crowsfoot" indicates the "many" side. A dashed line indicates optional participation, a solid line mandatory participation. Note that the minimum cardinality is denoted on the same side as in our approach, whereas the maximum cardinality is denoted on the opposite side. Relationships have two names, one in each direction ("role names"). Attributes are written inside the entity boxes. The symbol "#" indicates a primary key attribute (unique identification, see below), the symbol "*" a mandatory attribute ("not null"), and the symbol "o" an optional attribute (can be null).

Overview

Keys (1)

- A key of an entity-type E is an attribute which uniquely identifies the entities of this type.
- There may never be two different entities which have the same value for the key attribute.
- For example, the social security number is a key for persons: No two different persons can have the same SSN.

  I have heard that there are very rare cases where two persons have the same SSN. If this should be true, the SSN is no key.

Keys (2)

- It is possible to declare the combination of two or more attributes as a key: Then it is only forbidden that two entities agree in all of these attributes.

  Entities must be distinguishable by the value of at least one of the key attributes.

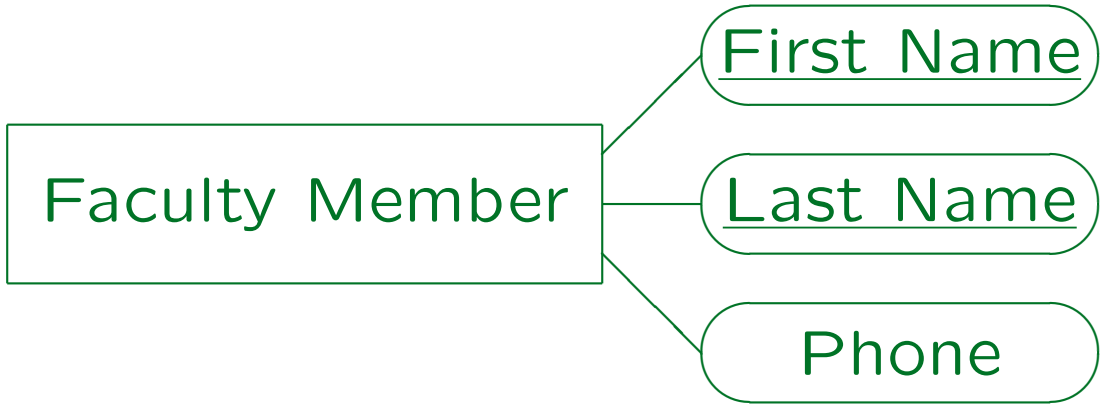
- E.g. using first name and last name together as key for faculty members, it is legal to have two professors with the same last name if their first names are different.

  It is also possible to have two faculty members with the same first name and different last names.

Keys (3)

Graphical Syntax:

- Keys are marked in ER-diagrams by underlining the attributes which form a key:



- Only entity types can have key attributes.

  Keys cannot be declared for relationships (but cardinality specifications are something similar to keys for relationships).

Keys (4)

Specification in Logic:

- Of course, key constraints can be specified as logical formulas. For instance, the key "First_Name, Last_Name" of "Faculty Member" means:

$$\forall \texttt{faculty\_member X, faculty\_member Y:}$$
$$\texttt{X.first\_name} = \texttt{Y.first\_name} \wedge$$
$$\texttt{X.last\_name} = \texttt{Y.last\_name} \rightarrow \texttt{X} = \texttt{Y}.$$

- Equivalent formulation:

$$\neg\exists \texttt{faculty\_member X, faculty\_member Y:}$$
$$\texttt{X.first\_name} = \texttt{Y.first\_name} \wedge$$
$$\texttt{X.last\_name} = \texttt{Y.last\_name} \wedge \texttt{X} \neq \texttt{Y}.$$

Keys (5)

Implication Between Key Constraints:

- Key constraints become weaker (i.e. less restrictive, more DB states are valid) if attributes are added.
- E.g. consider a department with professors Stefan Posch, Nina Brass, Stefan Brass. This state
  - ◇ violates the key constraint for First_Name,

    There are two "Stefans".
  - ◇ violates the key constraint for Last_Name,
  - ◇ satisfies the key constraint for the combination of First_Name, Last_Name.

Keys (6)

Minimality of Keys:

- If one has declared a key, e.g. SSN, then any superset of it (e.g. SSN together with Last_Name) automatically is a unique identification.

  If there cannot be two professors with the same SSN, there certainly cannot be two professors that have the same SSN and the same last name.

- The usual definition of a key requires that the set of key attributes $\{A_1, \ldots, A_k\}$ is minimal.

  Since the unique identification property automatically holds for supersets, "nonminimal keys" are indeed not interesting.

Keys (7)

Minimality of Keys, Continued:

- The minimality requirement means that we cannot leave out any of the key attributes without destroying the property of unique identification.
- However, in this definition, a key is not only
  - ◇ a constraint, which excludes invalid DB states,
  - ◇ but also an assertion about the real world, that certain states are possible.
- In the literature, a set of attributes that only satisfies the unique identification is called a "superkey".

Multiple Keys (1)

- An entity type may have more than one key, e.g.:
  - ◇ E.g. SSN is one key.
  - ◇ The combination of First_Name and Last_Name is another key.
- Both keys are minimal, because neither is a subset of the other. (None of the two implies the other.)

  It does not matter that the first key consists of only one attribute, whereas the second consists of two.

Multiple Keys (2)

- One of the keys is designated as the "primary key".
- Other keys are called "alternate/secondary keys".

  SQL uses the keyword `UNIQUE` for alternate keys.

- The primary key should be a key that consists only of a single, short attribute and is never updated (if available).

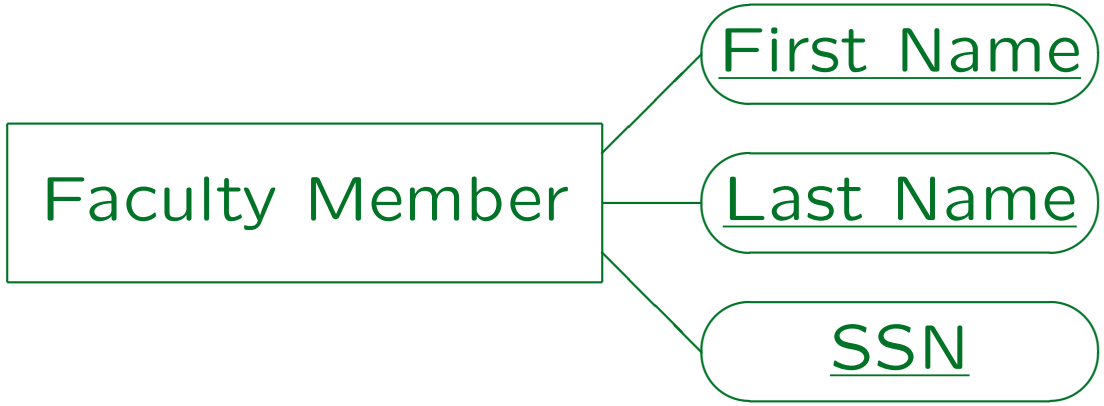
  After the translation into the relational model, the primary key is used in other tables that need to refer to entities of this type. In some systems, access via the primary key might be especially fast. Otherwise, the selection of the primary key is more or less arbitrary.

Multiple Keys (3)

- The graphical syntax allows only specification of a single key for each entity type (the primary key).

  In CASE tools, alternative keys can be stored. If the diagrams are drawn by hand, one should mention the alternative key in prosa.

- For instance, this would mean that there is only a single key consisting of all three attributes:



Are Keys Necessary?

- The ER-model does not require declaring a key for each entity-type (entities have an object identity).
- However, translating an ER-schema into the relational model requires a key for every entity type.

  For "weak entity types" (see below), the "key" contains a relationship (this is an extension of the notion of a key as defined above).

- If there is no natural key, identifying numbers can be added (sometimes called "surrogate key").
- CASE tools like Oracle Designer do this automatically when no key is declared.

Natural vs. Artificial Keys (1)

- Artificial keys are simple, but using only artificial keys also has disadvantages.

  Things like "order number" or "course number" that are already used in the real world are on the border between natural and artifical.

- The selection of a natural key (not an artificial identification number) is quite difficult.

  One of Murphy's Laws: There are always exceptions.

- Often, thinking about a key helps to understand the real meaning of a concept better.

  E.g. course offering in a term vs. entry in the course catalog (topic).

Natural vs. Artificial Keys (2)

- Even if an artificial number is used, think about the identification mechanisms used in the application programs.

  It is dangerous if different application programs use different identification mechanisms for the same thing.

- If the non-presence in the database is used to make statements about other objects in the real world (e.g. German "Schufa": "bad credit" database), all these objects must be uniquely identified.

  The objects about which information is stored might violate the key although the subset in the database satisfies it.

Natural vs. Artificial Keys (3)

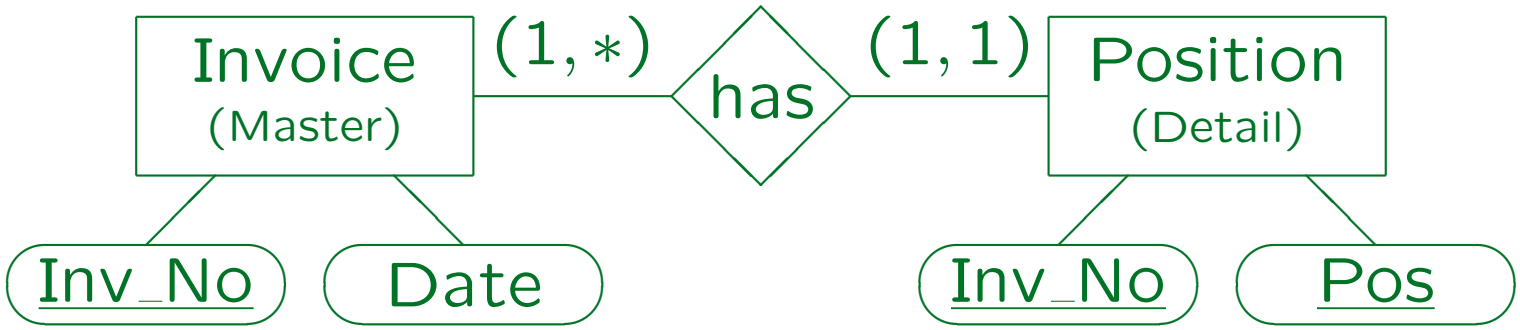
- The purpose of keys is not only the identification of entities.
- Keys should also help to avoid duplicates in the database. Artificial keys do not have this function.
- Often, duplicates are not exact copies: For instance, other shorthands or capitalization is used.

  Of course, the pure concept of a key then does not help. Furthermore, a constraint is not really needed — some warning would suffice.

- Think about possibilities for detecting duplicates before writing application programs.

Weak Entities (1)

- An entity may describe a kind of "detail" that cannot exist without a "master" or "owner" entity.
- Then there is a relationship with a (1,1) cardinality pointing to the owner.
- In addition the key of the owner entity is inherited and becomes part of the key of the detail entity:



Weak Entities (2)

- Without a specific construct for this situation, the following constraint would be required:
  - ◇ If two entities are connected via "has",
  - ◇ then their attribute "Inv_No" has the same value.

    E.g. invoice 12 cannot have position 2 in invoice 36 as detail.
- Such constraints occur when an entity does not have a key by itself, but is only unique in the context of some other entity.

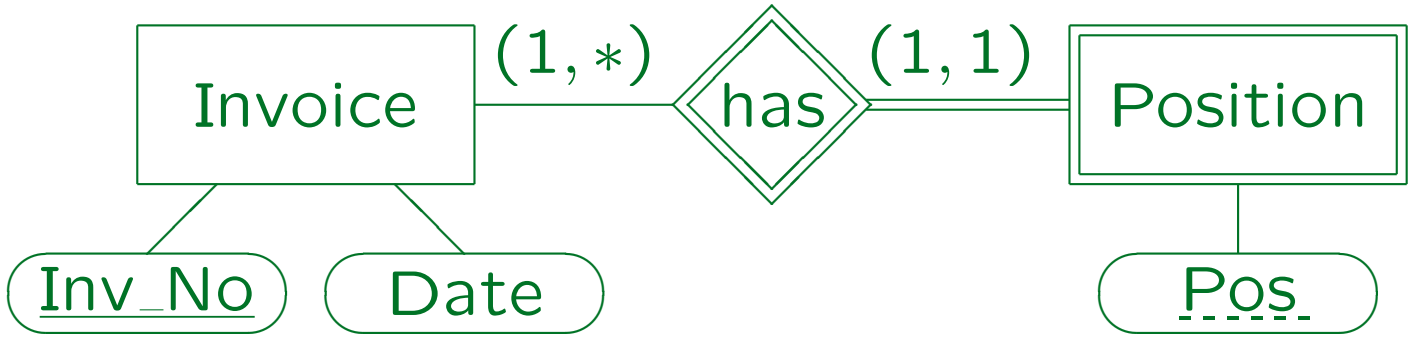
  I.e. it must "borrow" a key attribute of a related entity.

Weak Entities (3)

- In such cases there are always composed keys:
  - ◇ A classroom is identified by a building and a room number.
  - ◇ A subexercise is identified by the exercise number (e.g. 1) and a letter (e.g. a).
  - ◇ A web page is identified by a web server and a path on that server.
- There is also an existence dependency: If the building is pulled down, the rooms in it automatically disappear.

Weak Entities (4)

- Weak entities were introduced for this situation.
- They are marked by using double lines for their box, the connecting line, and the relationship diamond:



- Only the extension to the borrowed key is shown. Since it is only a partial key, it is dashed underlined.

Weak Entities (5)

- So the real key of the weak entity consists of the key attribute of the "owner" entity type (automatically inherited, not explicitly shown) plus the dashed-underlined partial key.
- Another way to look at this is that here the relationship contributes to the identification of the entity, whereas usually only attributes are used for this purpose (in a key).

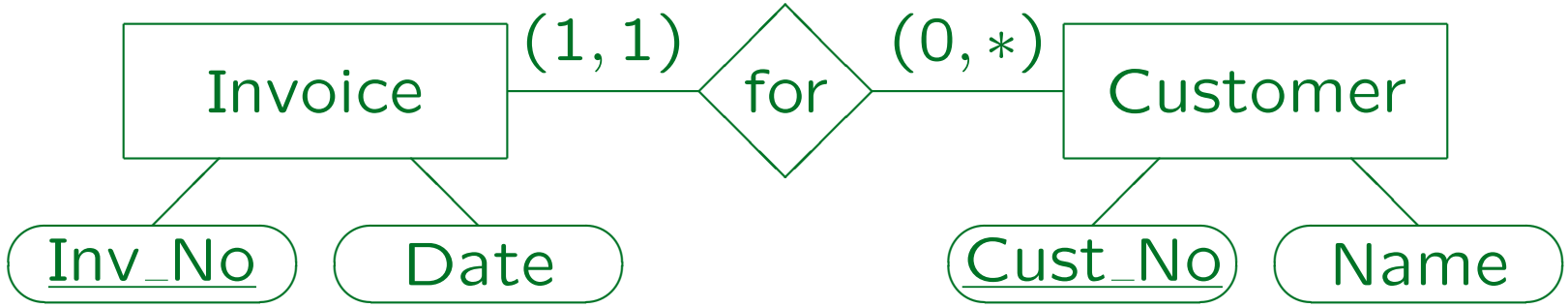
The double line can be understood as an "underlined line".

Weak Entities (6)

- The cardinality $(1,1)$ is implied by the weak entity construct. It does not have to be specified explicitly.

  Other relationships cannot be used: There would be no unique owner entity from which the key value can be inherited.

- The $(1,1)$ cardinality does not automatically mean "weak entity". The entity can still have a key of its own:



Weak Entities (7)

- A weak entity is needed only when the key of one entity contains the key of a related entity.

  A weak entity must add further key attributes to the key attributes inherited from its owner. If the keys of the two entity types would be the same, a specialization should be used (→ Course on DB Design).

- Sometimes the master/owner entity is called "parent entity", and the dependent weak entity is called "child entity".
- Entities with their own key (non-weak entities) are called regular/strong entities.

Weak Entities (8)

- Weak entities are normal entities except that their key is constructed in a special way.
- Thus, weak entities can have normal relationships besides the one via which the key is inherited:

- Weak entities can themselves be owner entities for other weak entities.

  There can be an entire hierarchy of parent-child relationships (e.g. "grandchildren"). But cycles are forbidden.

Weak Entities (9)

Exercise:

- Model a set of online quizzes (multiple-choice tests e.g. available on a course webpage).
- Each quiz is identified by a title, each question within a quiz by a number, and each answer to a given question by a letter.

  For each question and answer the text must be stored, and answers must be classified into correct and incorrect ones.

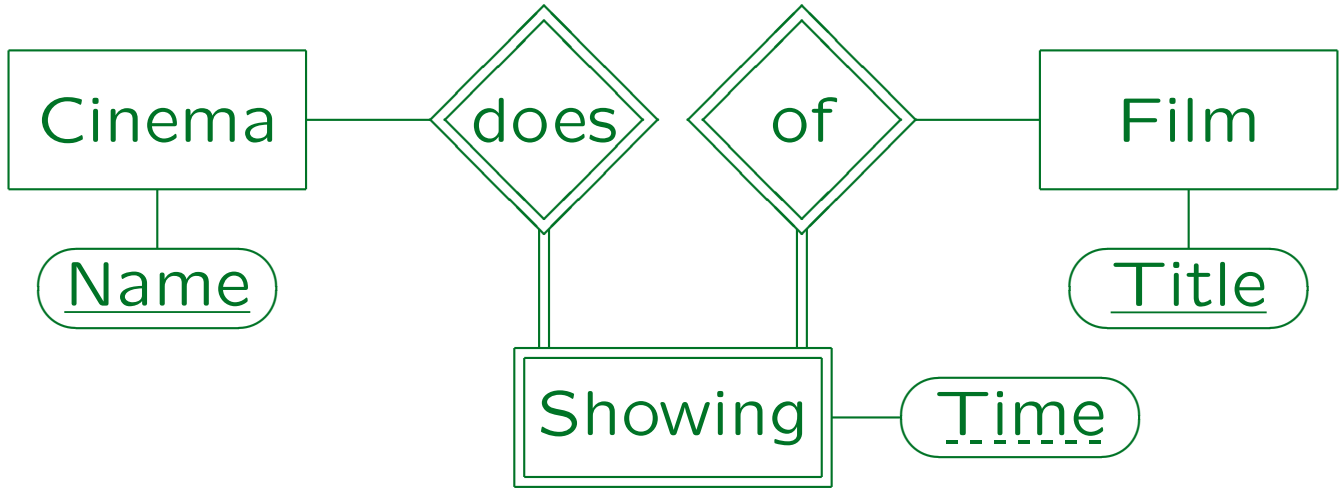
- What is the complete key of each of the three entity types?

Association Entities (1)

- It is somtimes necessary to turn a relationship into an entity, e.g. because:
  - ◇ Ternary relationships are excluded in this course.
  - ◇ Many CASE-tools do not support relationship-attributes.
  - ◇ Some style guides suggest to replace many-to-many relationships in this way by entities.
  - ◇ A relationship between a relationship and some entity type might be needed.
  - ◇ There is a multiple-valued relationship attribute.

Association Entities (2)

- The relationship is then turned into a weak entity. It inherits the keys of the entity types that participate in the original relationship.
- Weak entities can have multiple parents/owners:



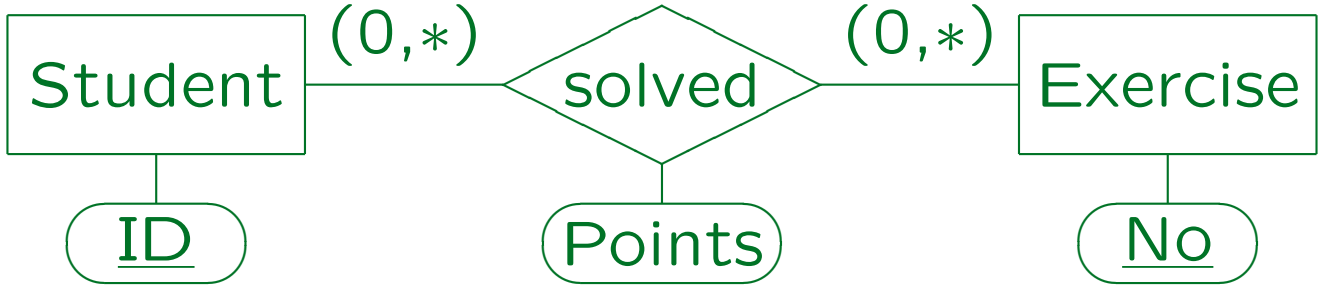
Association Entities (3)

- In the above example, the real key of the weak entity "Showing" consists of
  - ◇ "Name" (inherited from "Cinema"),
  - ◇ "Title" (inherited from "Film"), and
  - ◇ "Time".
- Weak entities types with several parents/owners are sometimes called "association entities".

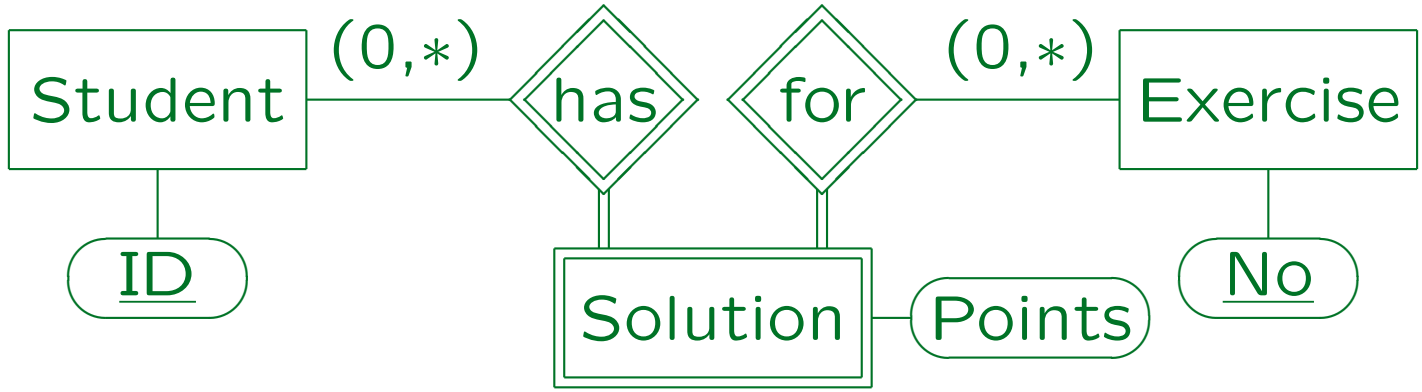
  If two parents have key attributes with the same name, (at least) one must be implicitly renamed.
- However, "weak entity" is equally ok.

Association Entities (4)

- Relationship:



- Association Entity (completely equivalent):



Graphical Syntax (1)

- All rules of Slide 3-24 to 3-26 still hold.
- The label of ovals connected to boxes may be
  - ◇ underlined (if the box has a single border) or
  - ◇ dash-underlined (if the box has a double border).

  These are the only cases where underlining is permitted.
- Lines connecting a box with a diamond may be labelled with a pair of numbers (n, m), where m may be "$*$". If m is not $*$, then $n \leq m$ must hold.

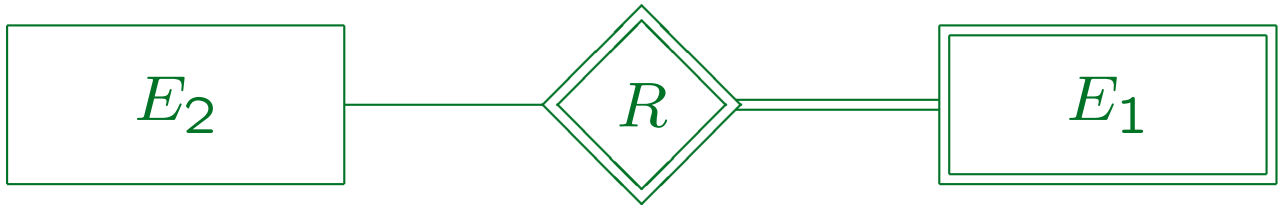
  "$*$" is not permitted in the first position. "0" makes no sense in the second position.

Graphical Syntax (2)

- Boxes may get a double border. In this case, at least one of the lines connecting it with a diamond must be double.
- Only lines connecting a box with a diamond may be double. In this case, both the box and the diamond must have a double border. The line on the other side of the diamond must be single.
- If a double line is labelled with a pair of numbers, it must be $(1,1)$.

Graphical Syntax (3)

- Consider the directed graph with
  - ◇ all boxes with double border as nodes,
  - ◇ an edge from E_1 to E_2 if E_1 is connected with a double line to a diamond that is also connected to E_2:



- This graph must not contain cycles.

Overview

Proper Naming (1)

- Names should be self-documenting.

  The correspondence to the real world must be clear. If necessary, attach comments, explanations, example data.

- Names should not be too long.

  Names longer than 15–20 characters become unwieldy.

- Choosing good names needs some thought.
  But the invested time will later pay off.

  Discussing the names with other people might help.

- Whatever you do, try to be consistent!

  Use abbreviations consistently. Use "_" consistently. In a team, all members should use the same style.

Proper Naming (2)

Entity-Types:

- Usually, nouns are used for entity-types.
- I prefer the singular form (names single entities).

  Some authors use plural forms. After the translation into the relational model, the plural form for tables seems first more approriate. But when tuple variables are declared in SQL, the singular form is more intuitive. Oracle Designer uses the singular form for entity types, and the plurarl form for the corresponding tables. Only be consistent.

- Don't use names like "Customer_Data": The goal is to create a model of the real world, not of the database.

Proper Naming (3)

Relationships:

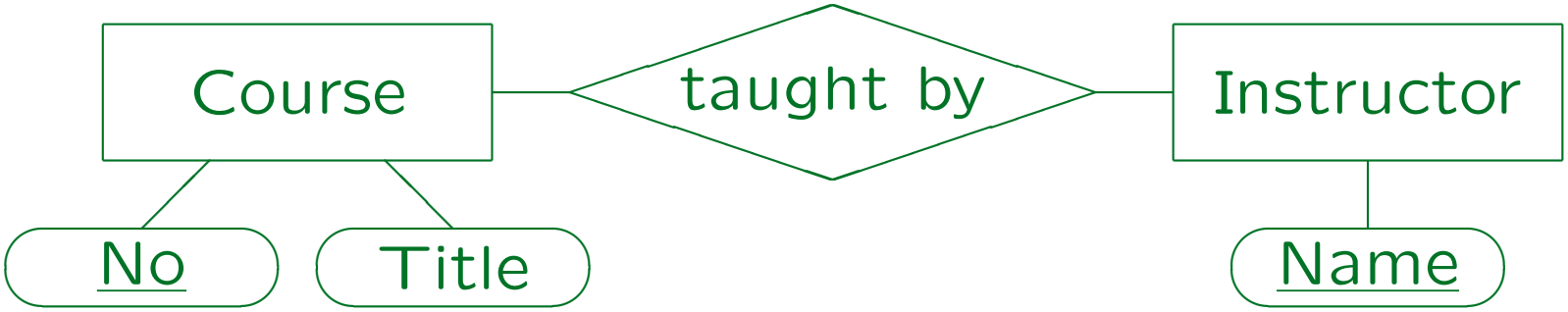
- Often, verbs are used for relationship-names.
- Relationship-names (e.g. "teaches") should read from top to bottom and from left to right.

  In many ER-model variants, relationships have two names: One in each direction. That solves the problem.

- Prepositions like "of" are quite unspecific. However, they can be combined with another word to make them clearer (e.g. "teacher of", "lies in").
- Some people use nouns like "Teaching_Assignment".

Attribute vs. Entity (1)

- Sometimes it is not clear whether a new entity-type is needed:



- One could add the name of the instructor as an attribute to the course:

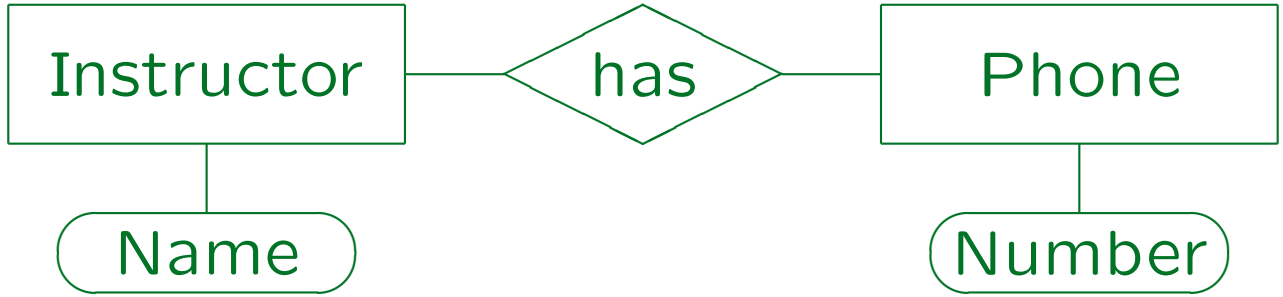
Attribute vs. Entity (2)

- Advantages of solution with "Instructor" entity:
  - ◇ If later more information has to be stored about instructors (e.g. phone number), only a small change of the schema is needed.
  - ◇ Because instructors are explicitly inserted, it is less likely that the same instructor is represented with different spellings (typing errors).

    Of course, this also depends on the application programs.
- But the solution with the "instructor name" attribute is simpler (also due to the implicit insertion).

Attribute vs. Entity (3)

- In principle, every attribute can be moved to an entity-type of its own:



- This design is only interesting for the phone company. Otherwise, it is unnecessarily complicated.
- Avoid entities which are only data type values.

Attribute vs. Relationship

- In principle, a relationship can be represented implicitly by attributes with the same value:

- This is not right in the ER-model. Relationships should be explicitly shown.

  In the relational model, it would be right. There is no explicit "relationship" construct in the relational model. But using such "foreign keys" in Entity-Relationship-schemas is a real mistake.

Redundant Information (1)

- Redundant information in an ER-schema is bad, because it is an unnecessary complication during conceptual design.

  One must specify at least an integrity constraint which ensures that both copies of the information are always consistent.

- Redundant information may be introduced during physical design for performance reasons.
- Furthermore, one can later define views which contain derived information.

Redundant Information (2)

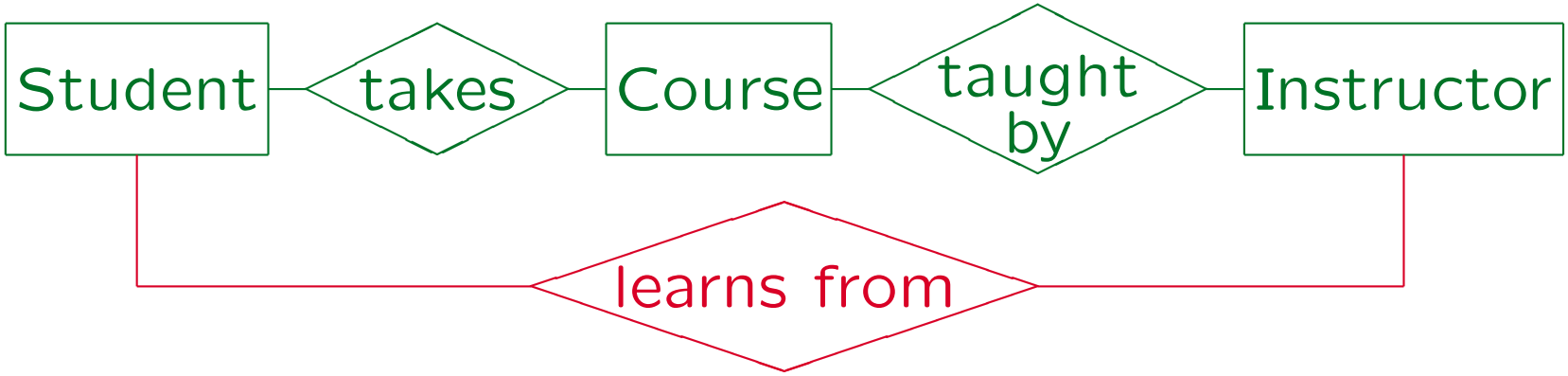
- One should not repeat attributes from other entity types which can be reached via a relationship:

- Many students who are already trained in relational database design make this mistake.

  There will be such a column when the schema is translated into the relational model (see below). But there are no relationships in the relational model. Having a relationship and an attribute pointing to the other entity is redundant.

Redundant Information (3)

- It is a mistake to define the composition of two relationships again as a relationship ("shortcut"):



- Different paths of relationships between two entity-types (cycles in the diagram) should be carefully inspected for possible redundancies.

  Of course, cycles are not always redundant or bad, but some dependency often exists (→ integrity constraint).