Part 9: 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. Kinds of Relationships (Cardinalities)
4. Keys, Weak Entities
5. Translation into the Relational Model

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 (or 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:
  - ◇ 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 consists of transforming this schema into the data model supported by the DBMS to be used (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.

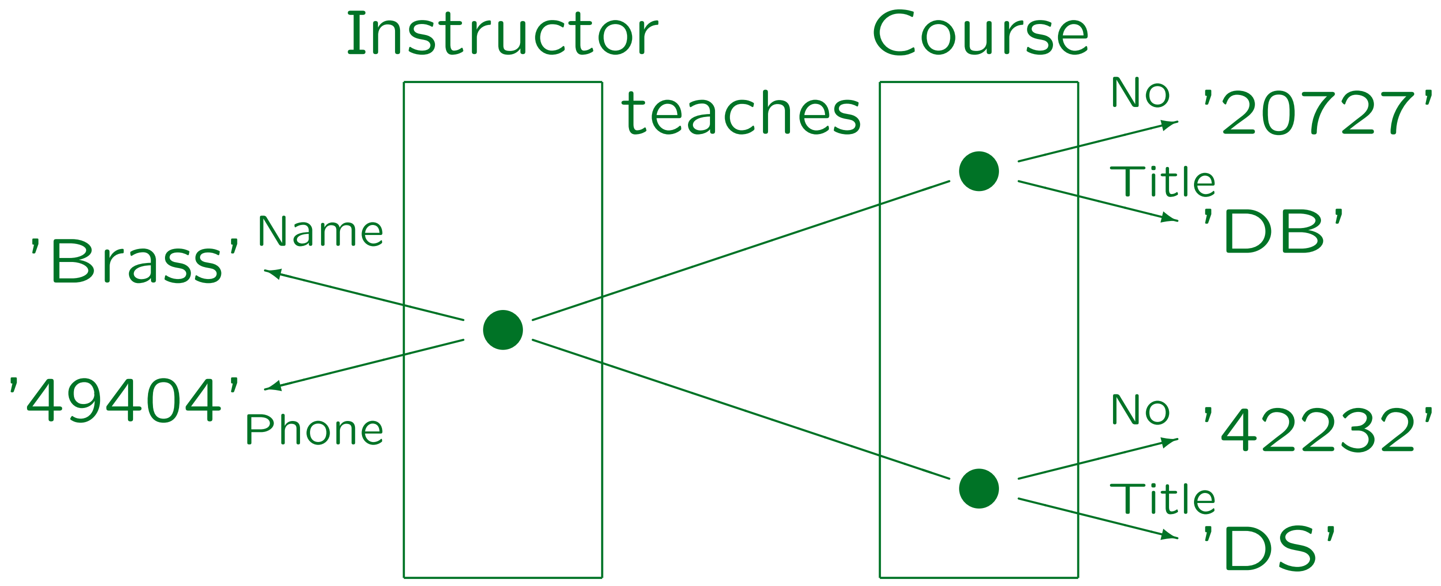
Example (1)

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.

    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.)

- 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

1. Database Design Overview
2. Basic ER-Constructs
3. Kinds of Relationships (Cardinalities)
4. Keys, Weak Entities
5. Translation into the Relational Model

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 (or a relationship).
- E.g. the title of this course is "Database Systems".
- 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 ("binary relationship").

  Some authors allow relationships involving more than two entities. My experience shows that this often leads to errors.

- E.g. I (a person) teach "Database Systems" (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 which has to be stored about them), i.e. entities which have the same attributes.
- E.g. all faculty members of this university.

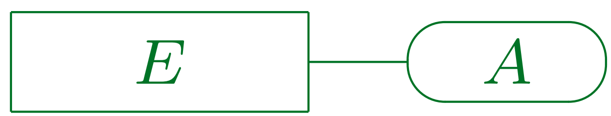
Relationship-Type:

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

ER-Diagrams (1)

- Entity-Type E:

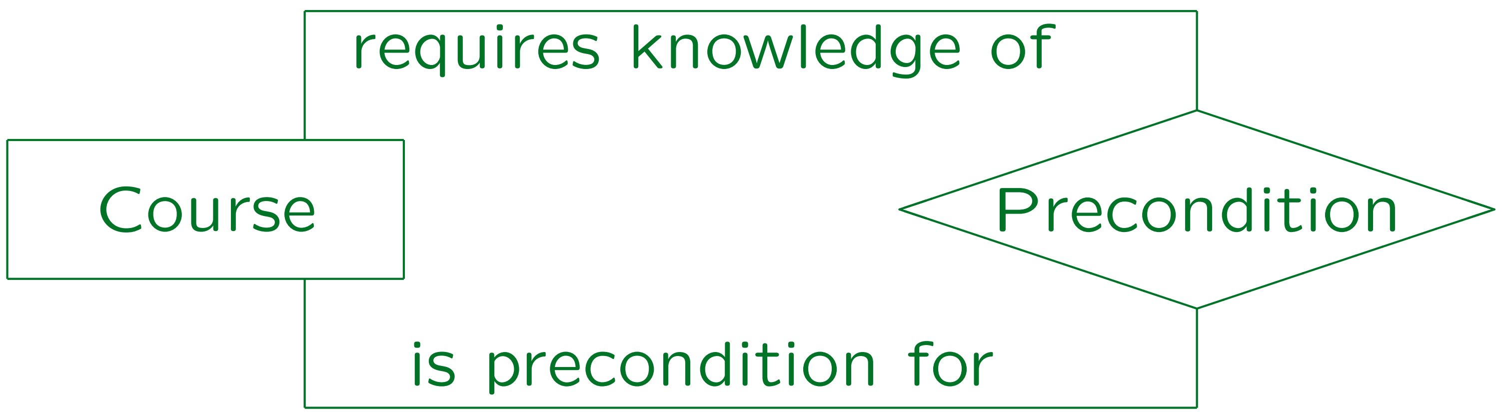
- Attribute A of Entity-Type E:



- 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”):

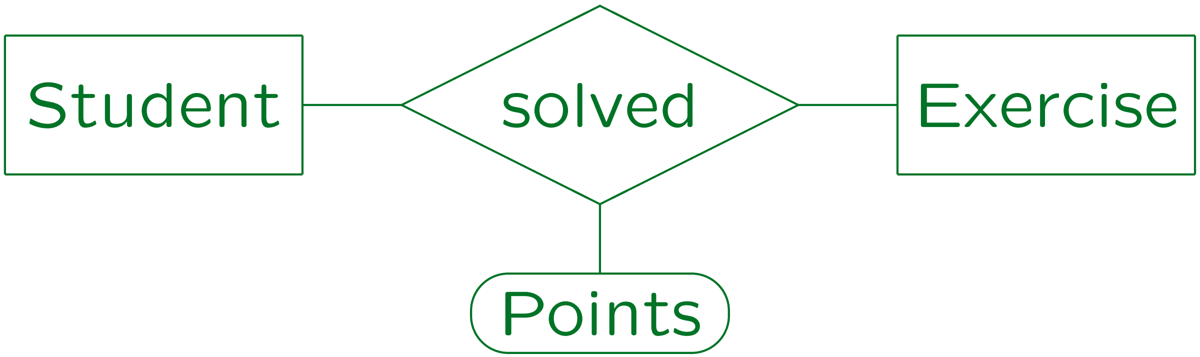


- In this case, “role names” must be attached to the connecting edges.

  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 must be unique in the entire diagram.

  I.e. you cannot have two boxes with the same label.

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

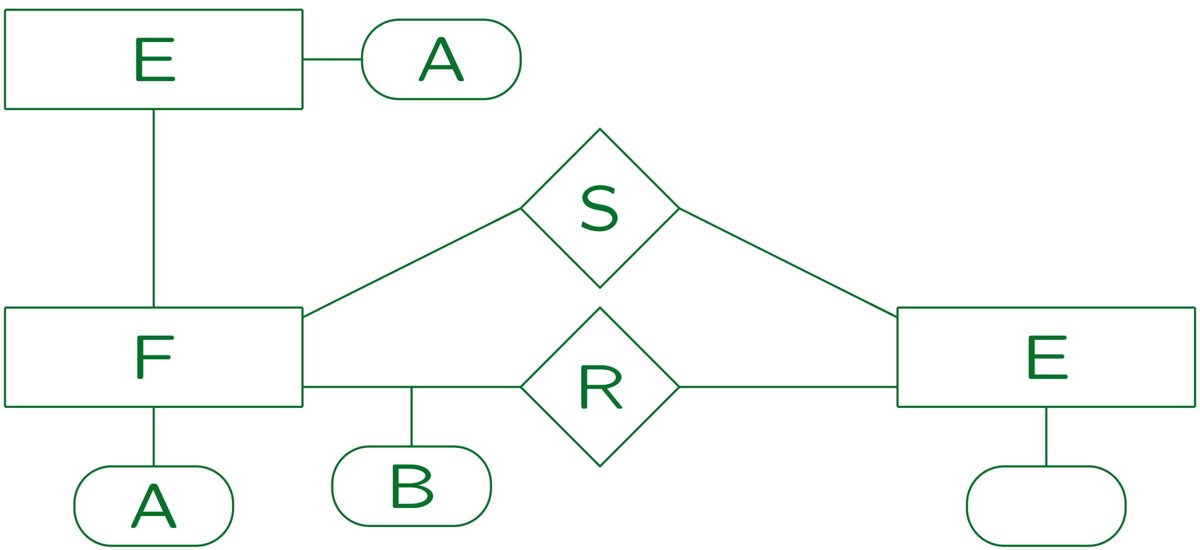
  I.e. you cannot have two ovals with the same name which are linked to the same box (or the same diamond).

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

  I.e. you cannot have 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 I interprets the symbols in the schema by defining

- a finite set $I(E)$ for every entity-type E,
- a mapping $I(A): I(E) \rightarrow val(D)$ 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 $I(R) \subseteq I(E_1) \times I(E_2)$ for every relationship R between entity types E_1 and E_2,
- a mapping $I(A): I(R) \rightarrow val(D)$ for every attribute A of a relationship R (D is the data type of A).

ER-Schemas: Semantics (2)

Example State:

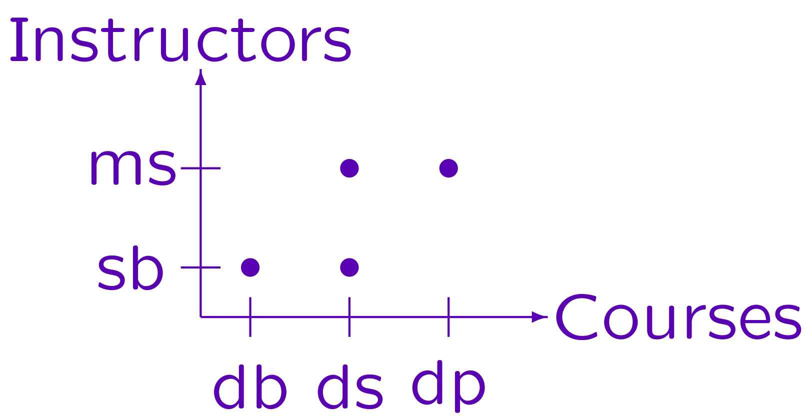
- $I(Instructor) = \{sb, ms\}$
- $I(Name) = f_1$, where
  $f_1(sb) =$ 'Brass', $f_1(ms) =$ 'Spring'.
- $I(Phone) = f_2$, where
  $f_2(sb) = 49404$, $f_2(ms) = 49429$.

ER-Schemas: Semantics (3)

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

ER-Schemas: Semantics (4)

- $I(teaches) = \{(sb, db),\ (sb, ds),\ (ms, ds),\ (ms, dp)\}$.
- 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 (5)

Consequences for Attributes:

- There are extensions, but the basic notion of a 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 (6)

Consequence for Relationships:

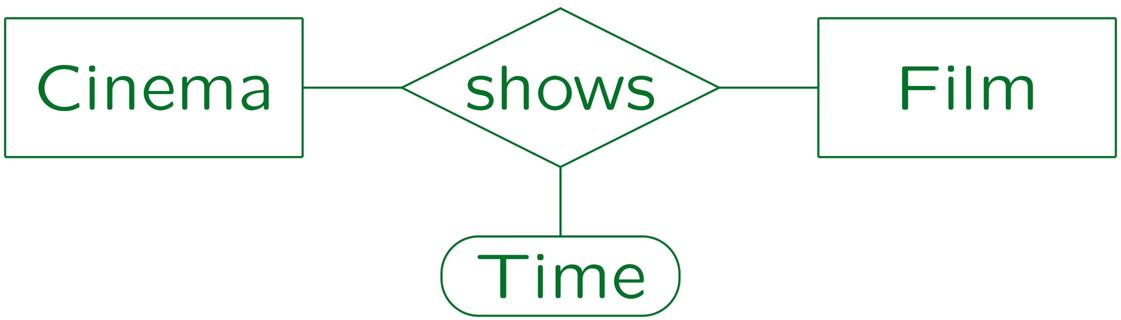
- A relationship is interpreted by a set of entity-pairs.
- A set either contains an element or does not contain it. It cannot be contained "two times".
- Therefore, a relationship either exists between two entities or it does not exist between these entities.

  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 (7)

Example/Exercise:

- Consider this schema:

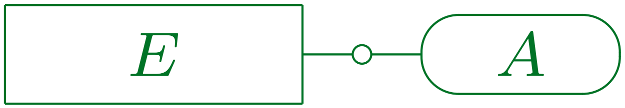


- Suppose a cinema shows the same film at 3pm and at 6pm.
- Can this information be stored in the given schema?

  ☐ 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:



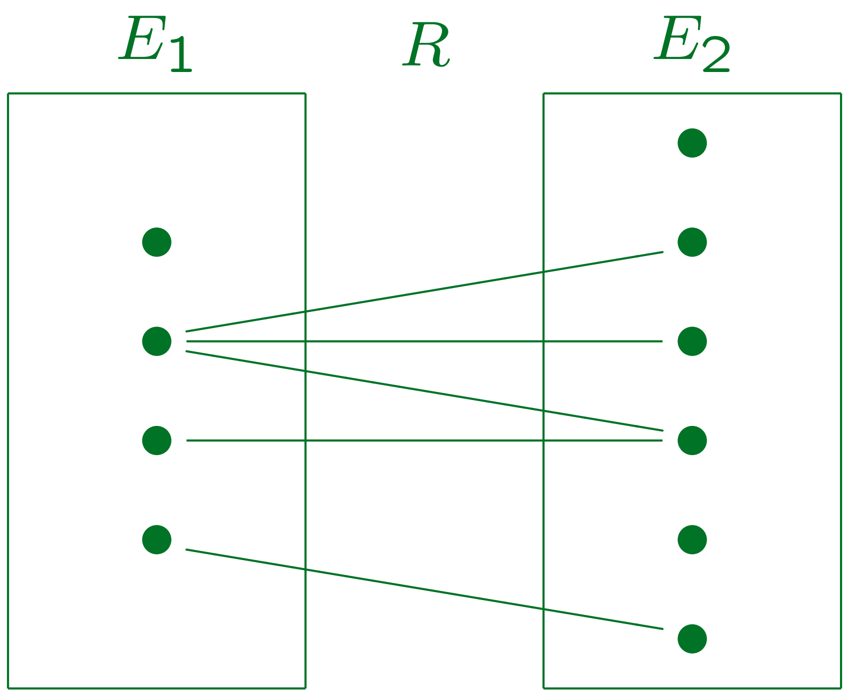
- The semantics of this attribute in a state I is a partial function from $I(E)$ (set of all E-entities) to $val(D)$ where D is the data type of A.

Overview

1. Database Design Overview
2. Basic ER-Constructs
3. Kinds of Relationships (Cardinalities)
4. Keys, Weak Entities
5. Translation into the Relational Model

Cardinalities (1)

General Relationship:

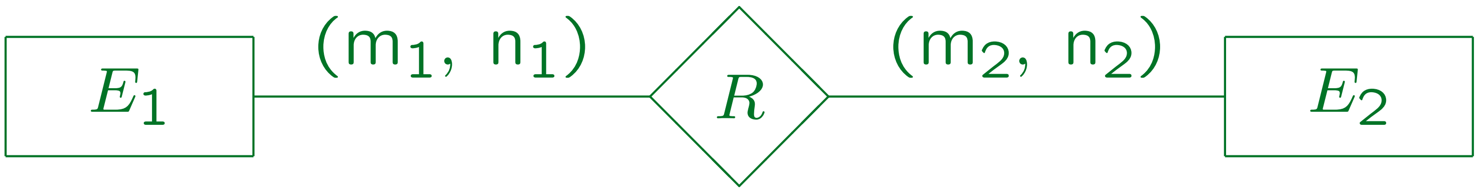


Cardinalities (2)

- 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 $e_1 \in I(E_1)$:

  $$\mathsf{m}_1 \;\le\; |\{e_2 \in I(E_2) \mid (e_1, e_2) \in I(R)\}| \;\le\; \mathsf{n}_1.$$

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

  $$\mathsf{m}_2 \;\le\; |\{e_1 \in I(E_1) \mid (e_1, e_2) \in I(R)\}| \;\le\; \mathsf{n}_2.$$

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 9-40, 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.

- 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 "many" side, but has maximum cardinality 1.
  - ◇ "Course" is the "one" side, but has maximum cardinality $*$.
- "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.

Alternative Notations (1)

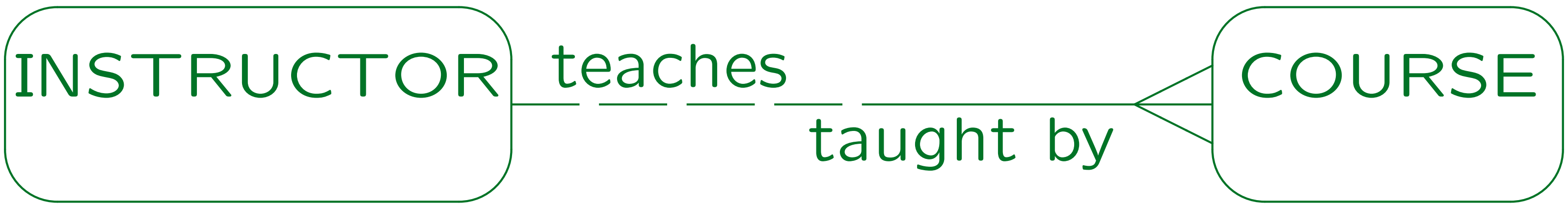
- There are many variants of the ER-Notation. Cardinalities are one point in which they differ quite significantly.
- E.g. one can use only “many to many” (N:M), “one to many” (1:N), and “one to one” (1:1).

- 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.
- Relationships have two names.

  One in each direction ("role names").

Overview

1. Database Design Overview
2. Basic ER-Constructs
3. Kinds of Relationships (Cardinalities)
4. Keys, Weak Entities
5. Translation into the Relational Model

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 these attributes.

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

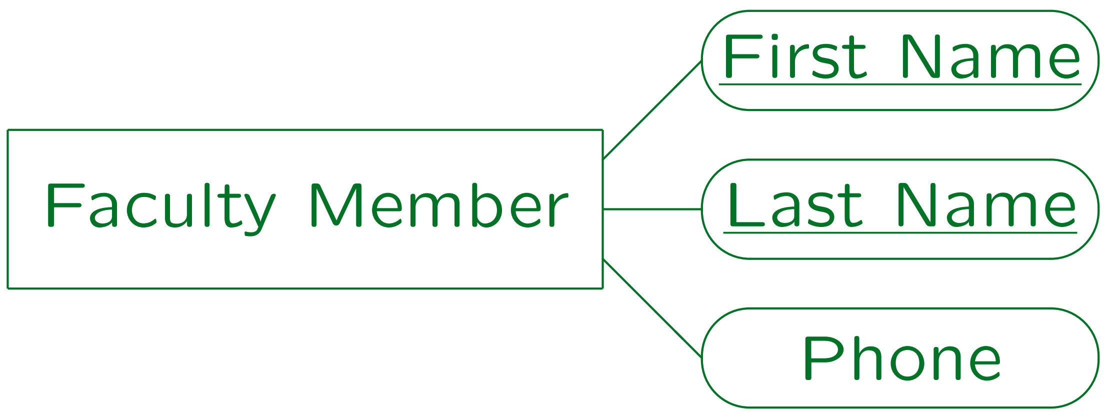
- E.g. using first name and last name together as key for faculty members, it is legal to have two professors with the same last name if their first names are different.

  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)

Multiple Keys:

- An entity type can have more than one key (e.g. if also the SSN is stored).
- The graphical syntax allows only specification of a single key for each entity type.

  If also the SSN is underlined, this means that the three attributes First Name, Last Name, SSN together form a key.

- Select one key as the "primary key".

  Such a unique "primary key" is needed for translation into the relational model. Select a key which consists only of a single, short attribute and does not change over time (if available).

Keys (5)

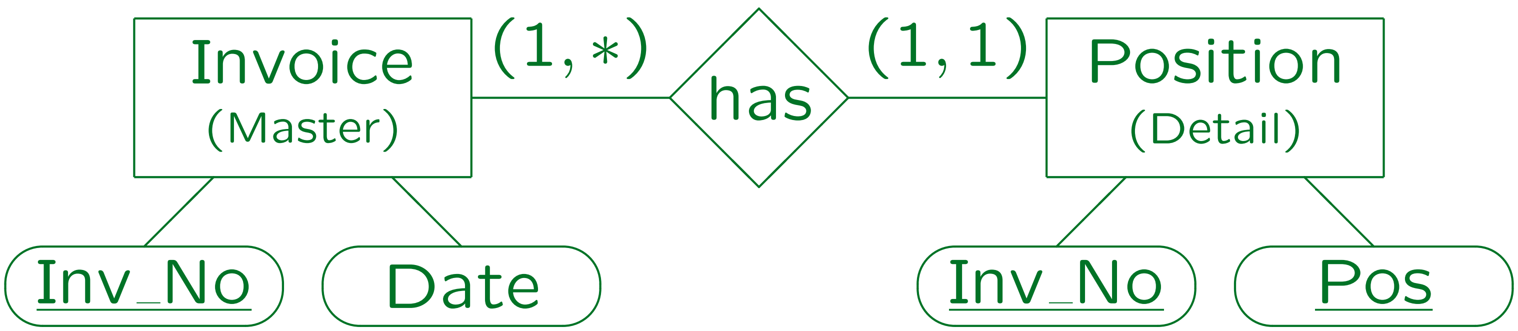
Importance of Keys:

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

  One extension to the basic ER-model are "weak entity types" which are identified in part by a relationship.
- If there is no natural key, identifying numbers can be added (e.g. "order number", "course number").

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.

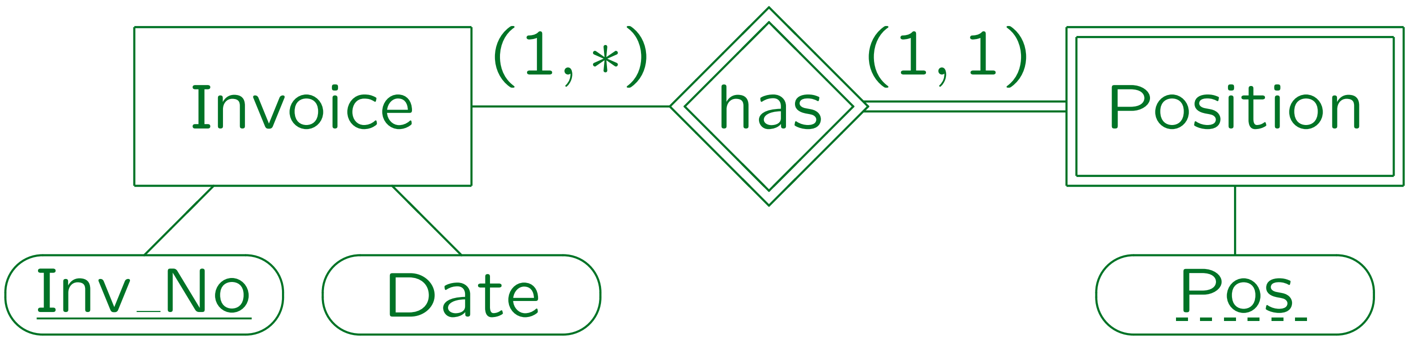
  I.e. it must "borrow" a key attribute of a related entity.

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).

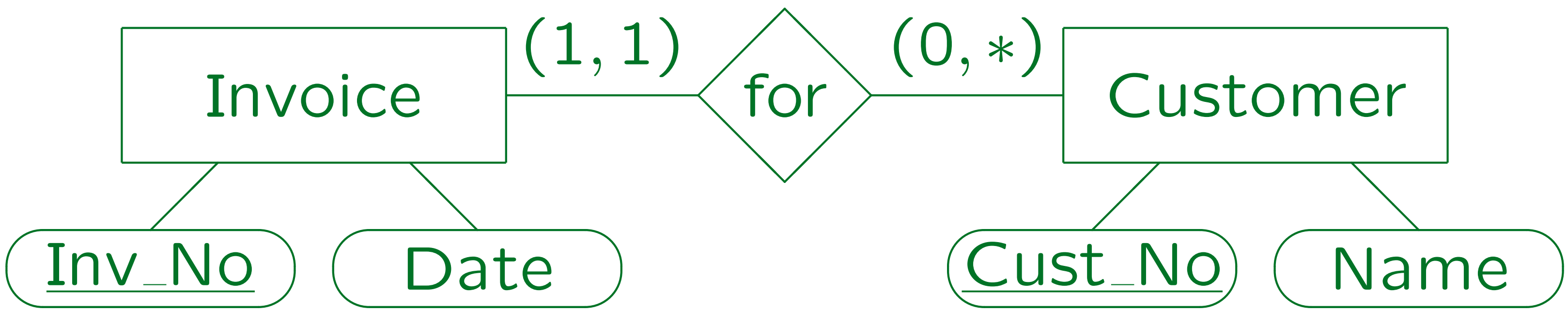
  The double line can be understood as an "underlined line".

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 of its owner. Otherwise use a specialization (→ below).

- 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.

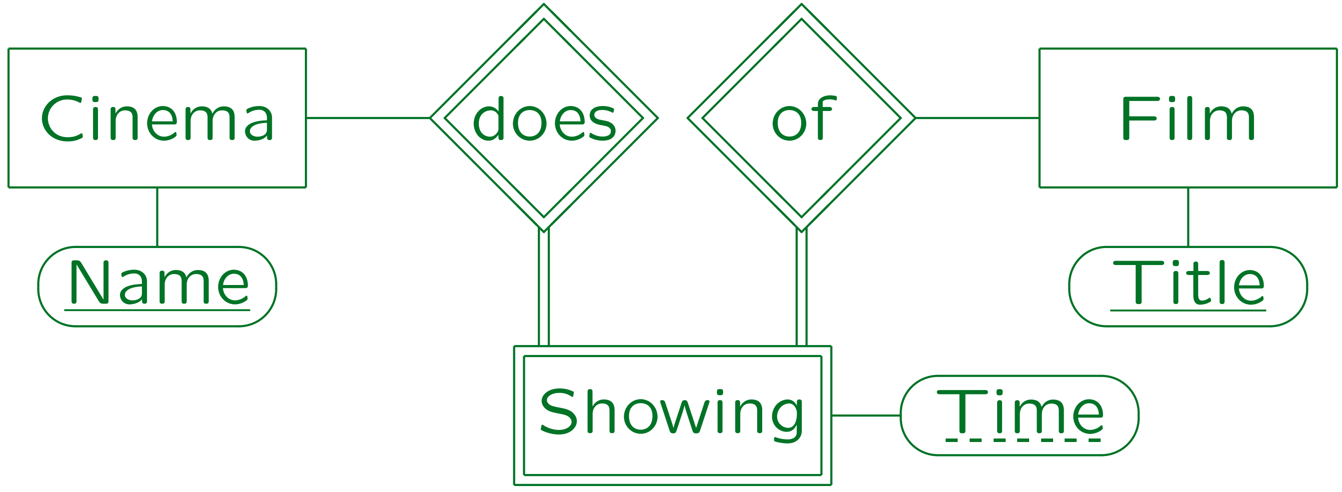
  For each question and answer the text must be stored, and answers must be classified into correct and incorrect ones.
- What is the complete key of each of the three entity types?

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:



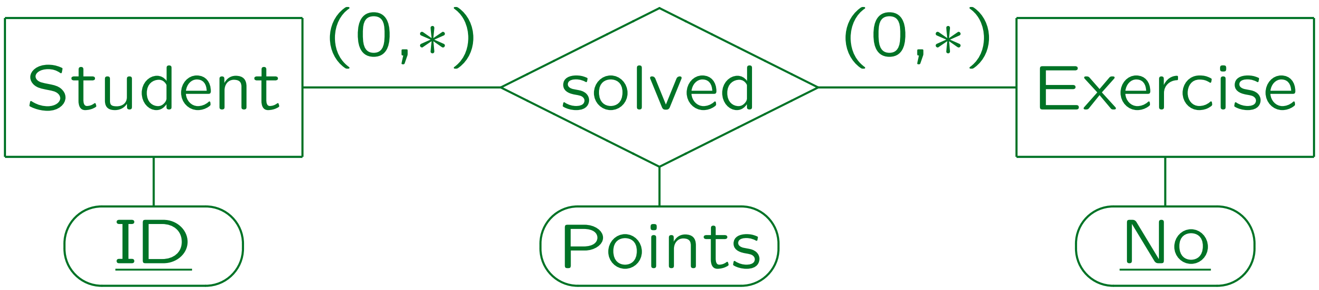
Association Entities (3)

- 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”.

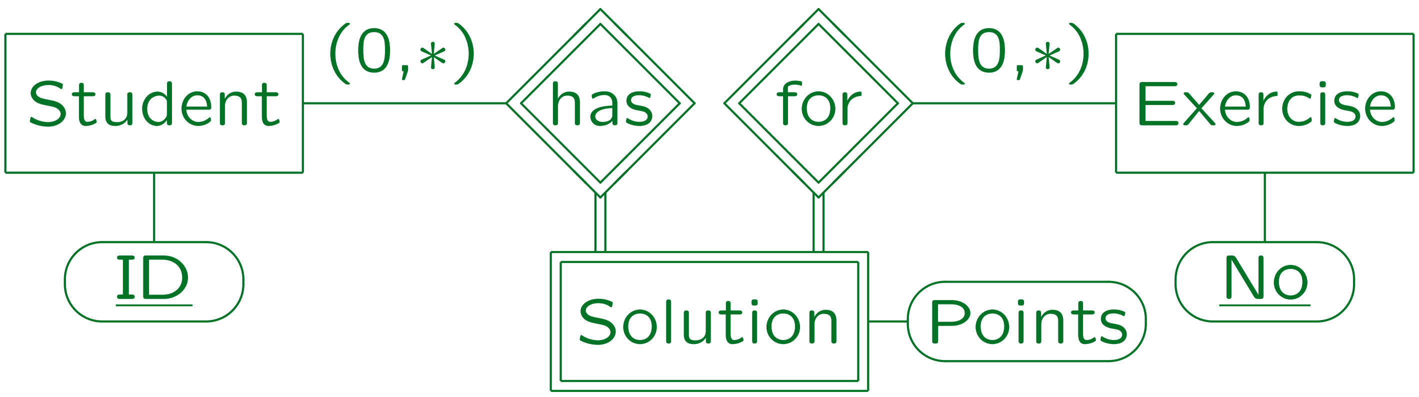
  If two parents have key attributes with the same name, (at least) one must be implicitly renamed.
- However, “weak entity” is equally ok.

Association Entities (4)

- Relationship:



- Association Entity (completely equivalent):



Overview

1. Database Design Overview

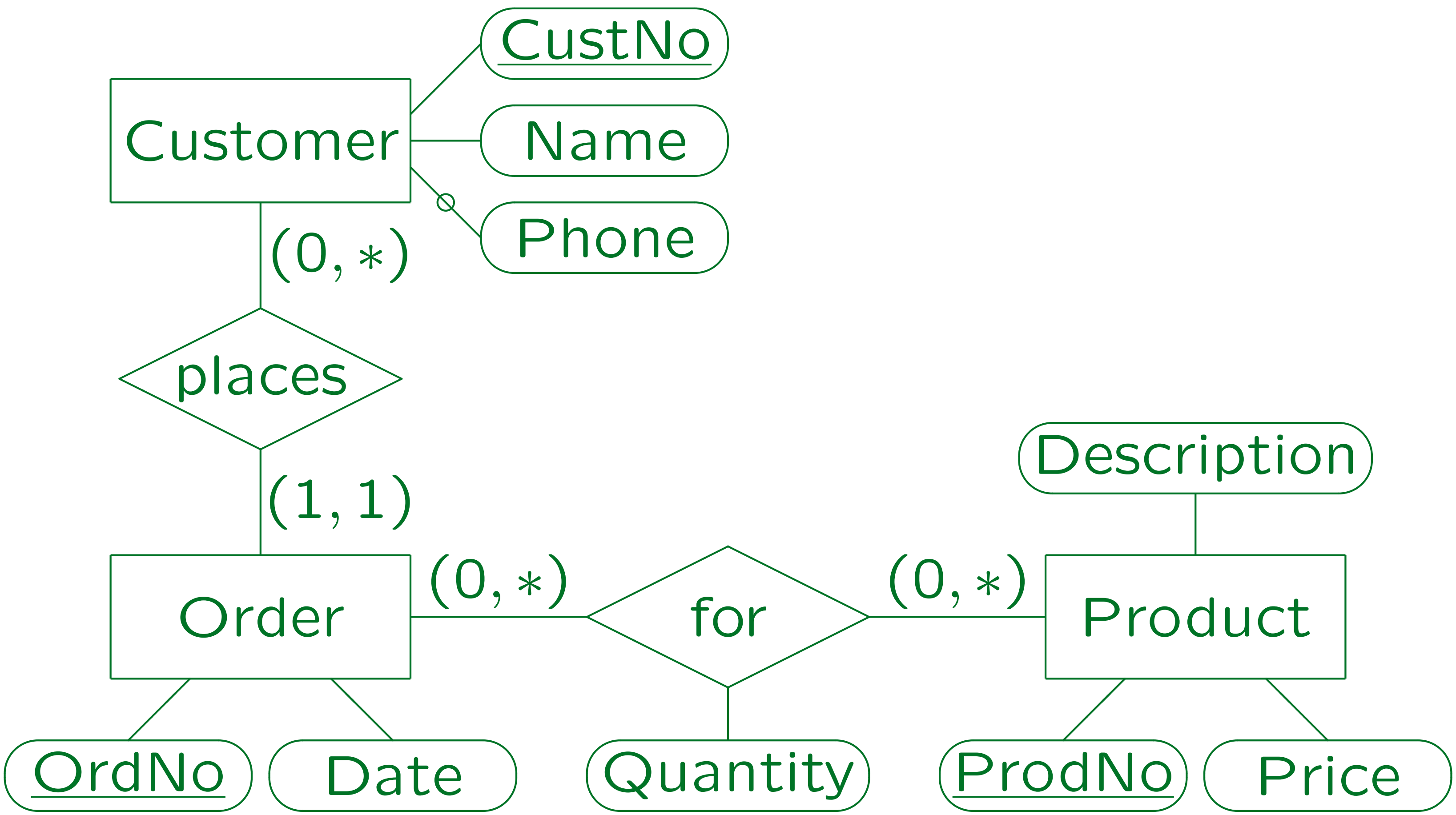
2. Basic ER-Constructs

3. Kinds of Relationships (Cardinalities)

4. Keys, Weak Entities

5. Translation into the Relational Model

Example


CustNo
Customer
Name
Phone
(0, ∗)
places
(1, 1)
Description
Order
(0, ∗)
for
(0, ∗)
Product
OrdNo
Date
Quantity
ProdNo
Price

Step 1: Entities (1)

- First create a table for each entity. The name of this table is the name of the entity type.

  Alternatively, the plural form can be used.

- The columns of this table are the attributes of the entity type.

  Optional attributes translate into columns allowing null values.

- The primary key of the table is the primary key of the entity type.

  If the entity type has no key, add an artificial key.

Step 1: Entities (2)

Customers		
CustNo	Name	Phone
10	Jones	624-9404
11	Smith	

Orders	
OrdNo	Date
200	2/15/00
201	2/16/00

Products		
ProdNo	Description	Price
1	Apple	0.50
2	Kiwi	0.25
3	Orange	0.60

Step 2: One-To-Many Rel. (1)

- If a relationship has the maximum cardinality 1 on one side, it is one-to-many. E.g. "places" is one-to-many from "Customer" to "Order".

  If it has maximum cardinality 1 on both sides, it is actually one-to-one (see below).

- In this case you add the key of the "one" side (Customer) as a column to the "many" side (Order).

- This column will be a foreign key referencing the row which corresponds to the related entity.

Step 2: One-To-Many Rel. (2)

- Result in the example:

  `Orders(OrdNo, Date, CustNo→Customers)`

Orders		
<u>OrdNo</u>	Date	CustNo
200	2/15/00	11
201	2/16/00	11

Customers		
<u>CustNo</u>	Name	Phone
10	Jones	624-9404
11	Smith	

Step 2: One-To-Many Rel. (3)

- If the minimum cardinality is 1 (as in this example), null values are not allowed for the new foreign key column.
- If the minimum cardinality should be 0, null values must be allowed for the foreign key column.

  It is null for entities not participating in the relationship.

Step 2: One-To-Many Rel. (4)

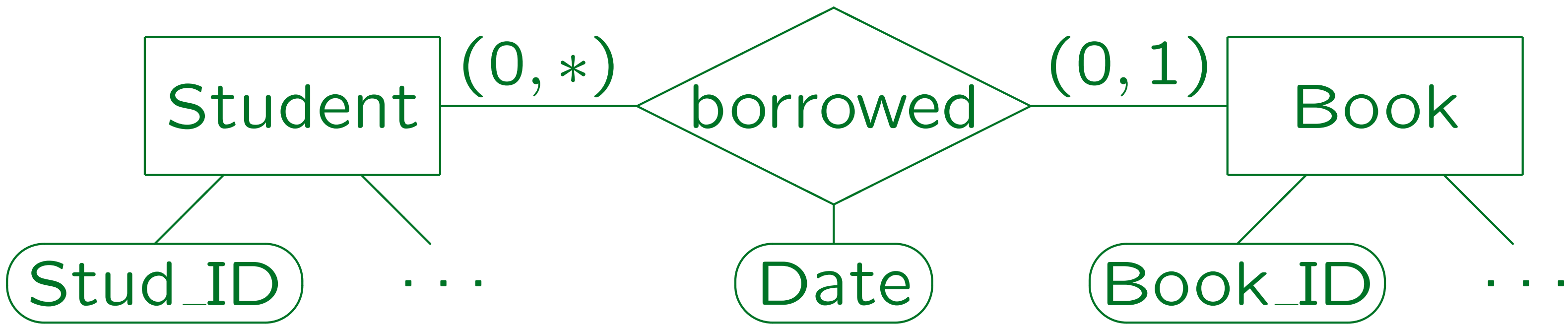
- The relationship name can be used in the column name, e.g.

  `Orders(OrdNo, Date, placed_by→Customers)`

- This excludes natural joins, but it is matter of style.
- Of course, all columns in the table must have unique names. If the added foreign key has the same name as an existing column, one or both must be renamed.

Step 2: Relationship Attributes

- Relationships can have attributes, e.g.:



- Such attributes are stored together with the pointer to the related entity:

  Books(<u>Book_ID</u>, ..., Stud_ID^o→Students, Date^o)

  "Stud_ID" and "Date" can be null, since not every book is borrowed, but they can either be both null or both not null (→ CHECK-constraint).

Step 2: A Variant

- One-to-Many Relationships with cardinality $(0, 1)$ can be translated into a table of their own:

  `borrowed_by(Book_ID→Books, Stud_ID→Students, Date)`

- The key values of the related entities are stored, plus attribute values of the relationship.

- The key attributes of the side with the $(0, 1)$ cardinality become the key of this relation.

  Every book can be borrowed only once at the same time.

- This does not work with the cardinality (1,1).

Step 3: Many-To-Many R. (1)

- A relationship is many-to-many if it has the maximum cardinality $*$ on both sides (e.g. "for").
- Many-to-many relationships become their own tables.
- The columns of this table are the keys of the participating entity types, which together form the key of this table.
- These columns are at the same time foreign keys, referencing the tables for the entity types.

Step 3: Many-To-Many R. (2)

- Relationship attributes are added as columns. They are not part of the key, e.g.

  for(<u>OrdNo</u>→Orders, <u>ProdNo</u>→Products, Quantity)

- Note that the key of "for" must really consist of both, "OrdNo" and "ProdNo".

  Since one order can request multiple products, "OrdNo" alone cannot be key. Since the same product may be subject of multiple orders, also "ProdNo" alone does not suffice.

- Tables can be renamed. E.g. "`Order_Details`" is a better table name than "`for`".

Step 3: Many-To-Many R. (3)

for

OrdNo	ProdNo	Quantity
200	1	1
200	2	1
201	1	5

Orders

OrdNo	Date	CustNo
200	2/15/00	11
201	2/16/00	11

Products

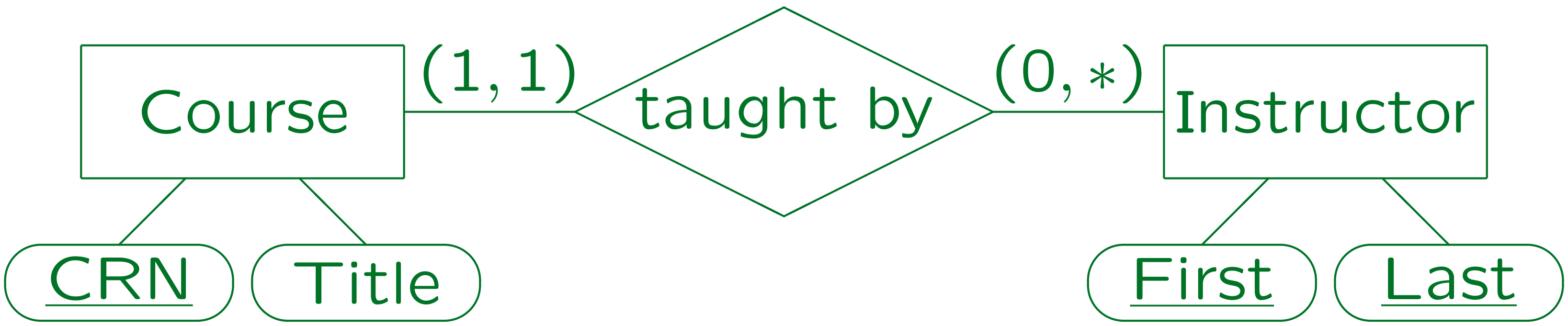
ProdNo	Description	Price
1	Apple	0.50
2	Kiwi	0.25
3	Orange	0.60

Step 3: Many-To-Many R. (4)

- Minimum cardinalities other than 0 for many-to-many relationships cannot be enforced by the standard constraints of the relational model.
- E.g. it would make sense to require that every order must be "for" at least one product. But this becomes a general constraint in the relational model.
- Of course, since this is important for the validity of the database state, one can specify the cardinality and later do checks in application programs.

  It only cannot be specified declaratively in the `CREATE TABLE`.

Composite Foreign Keys

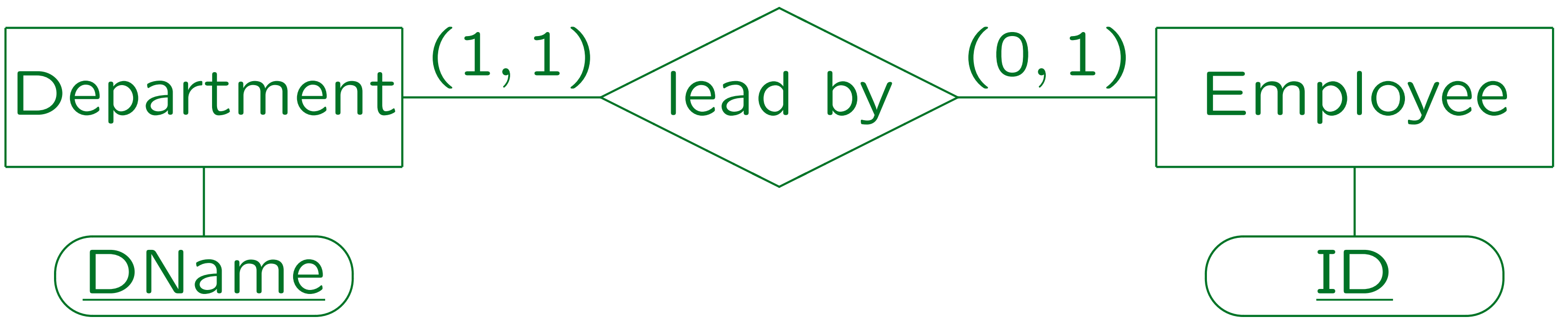


- A composite foreign key is used to reference a table with a composite key:

  Course(<u>CRN</u>, Title, (First, Last) → Instructor)

- If the minimum cardinality is 0, "First" and "Last" can be null, but only together.

Step 4: One-to-One Rel. (1)



- A relationship is one-to-one if it has maximum cardinality 1 on both sides.
- Basically, the translation is the same as for one-to-many relationships.

  However, there is an additional key constructed, see below.

Step 4: One-to-One Rel. (2)

- In this example, it is better to include the Employee key in the Department table, than vice versa, since Department has cardinality (1,1):

  Department(<u>DName</u>, . . . , Head → Employee)

- In this way, null values are avoided and the minimum cardinality 1 is enforced.

  If the department name is included in the Employee table, it can be null. In addition, a general constraint is required to ensure that every departments has a department head.

Step 4: One-to-One Rel. (3)

- “Head” is now also a key for the table “Department” (!), since an employee can be head of at most one department.
- It is only an alternative key, not part of the primary key.
- It enforces the maximum cardinality 1 on the Employee side.

Step 4: One-to-One Rel. (4)

- The key of any of the two tables can be included in the other table (as a possibly null foreign key).

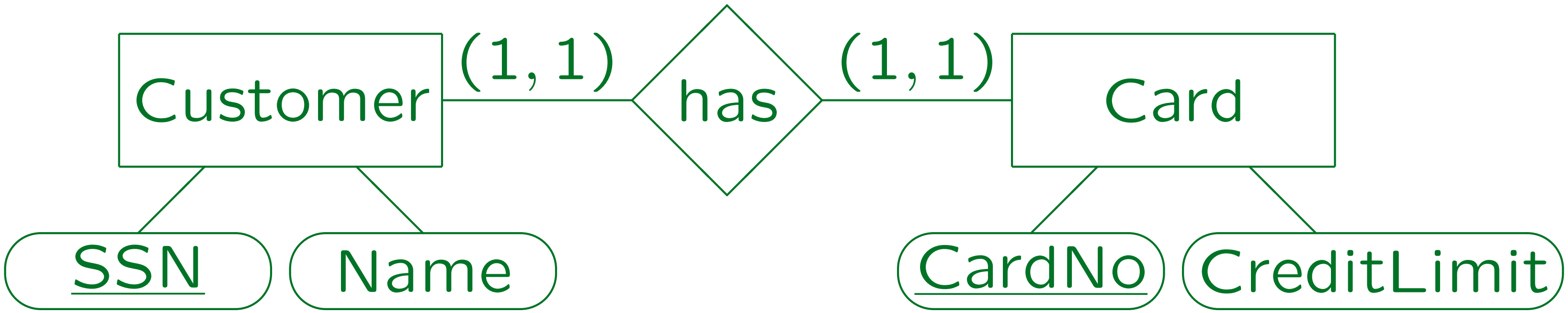
  However, it would be wrong to do both (redundancy).

- Or translate the relationship into a table on its own:

  Marriage(MName → Man, WName → Woman)

- Exercise: What is the key / keys?

Step 4: One-to-One Rel. (5)



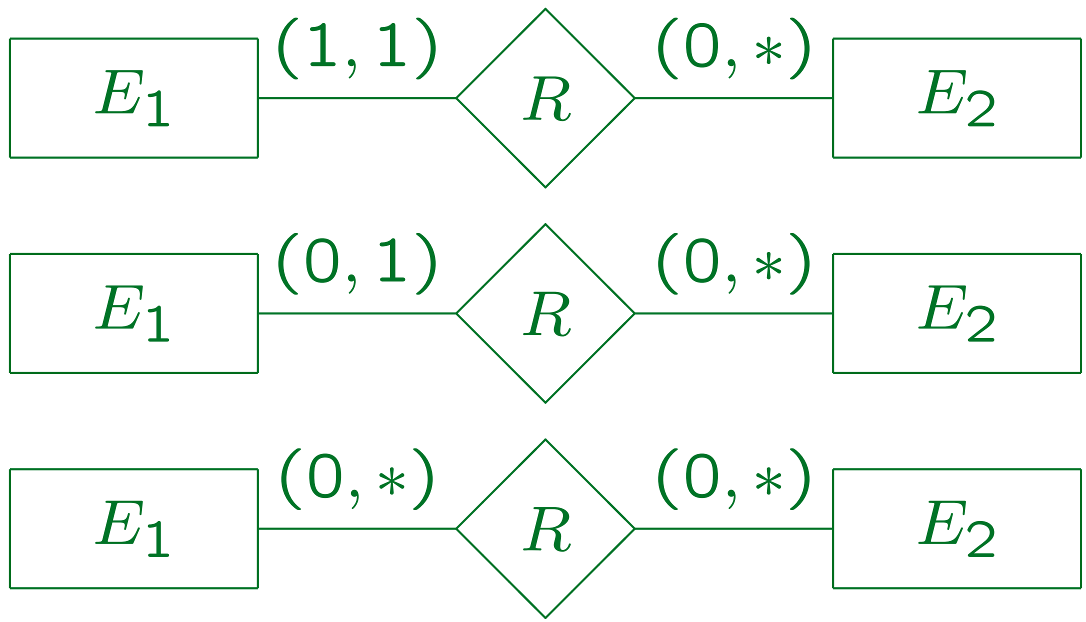
- In order to enforce the minimum cardinality 1 on both sides, the tables must be merged:

  CustomerCard(SSN, Name, CardNo, CreditLimit)

- SSN and CardNo are both keys. One is selected as primary key, the other is an alternative key.

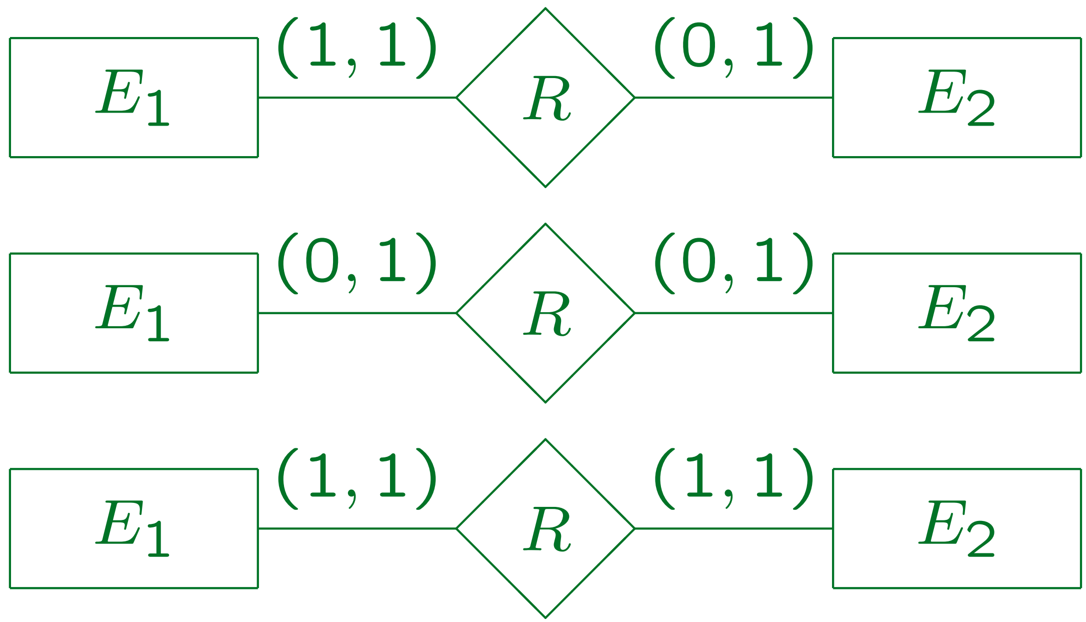
Limitations (1)

- The following cardinalities can be translated with the methods explained above (using only the standard constraints of the relational model):



Limitations (2)

- In addition, all kinds of one-to-one relationships can be handled:

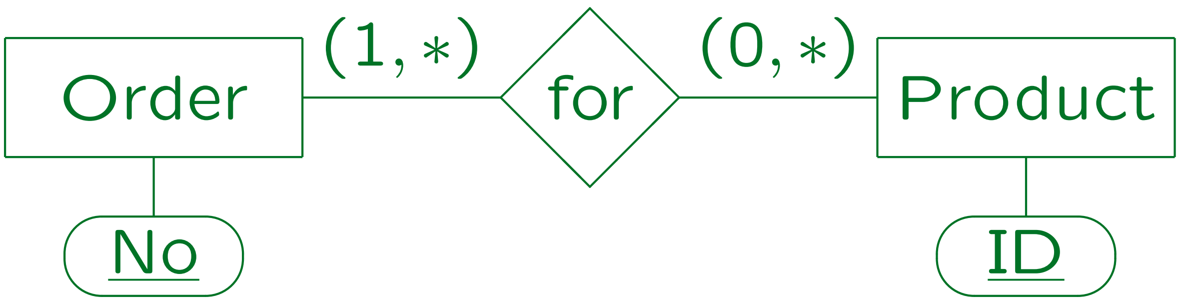


Limitations (3)

- If a relationship is none of these six cases, general constraints must be used, which can be implemented, e.g. via
  - ◇ Checks in application programs that are used to insert data.
  - ◇ Triggers, i.e. procedures stored in the database that are automatically executed e.g. for every tuple that is inserted or modified.
  - ◇ SQL queries that are executed from time to time and that print violations to the constraints.

Limitations (4)

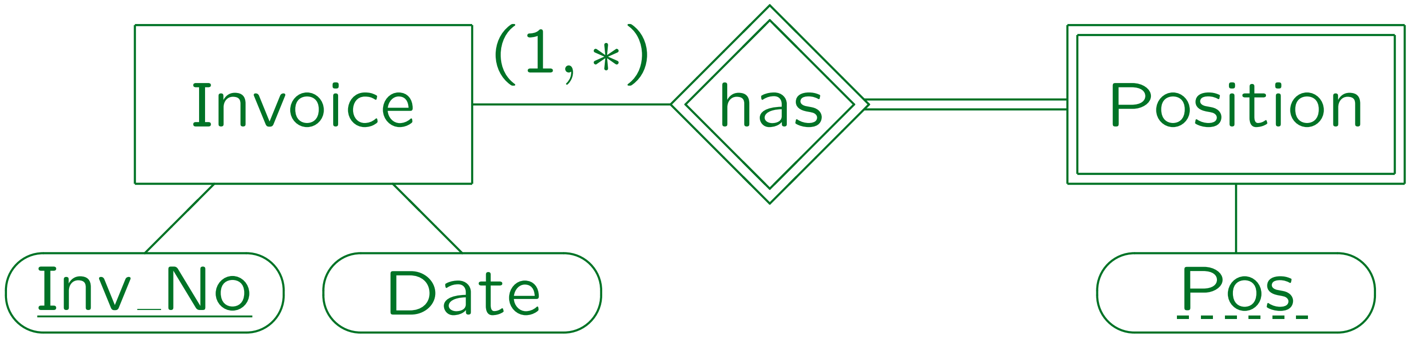
- In particular, the cardinality $(1, *)$ sometimes makes sense, e.g. every purchase order should be for at least one item:



- The minimum cardinality 1 cannot be enforced declaratively in current DBMS.

    One uses the same translation as for $(0, *)$ and documents/specifies a general constraint in addition.

Step 1B: Weak Entities



- When a weak entity is translated, the key attributes of the owner entity must be added as a foreign key:

  Position(<u>Inv_No</u> → Invoice, <u>Pos</u>, ...)

- This automatically implements the relationship.

  Such relationships must be ignored in Step 2.
  It makes sense to specify "`DELETE CASCADES`" for the foreign key.
  Note that there is no "dashed underlining" in the relational model.

Step 5: Check (1)

- At the end, check the generated tables to see whether they make sense.
- E.g. fill them with a few example rows.
- If a correct ER-schema is correctly translated into the relational model, one will get a correct relational schema.
- However, a by-hand translation can result in mistakes, and the ER-schema can contain hidden flaws.

Step 5: Check (2)

- Sometimes tables are redundant and can be deleted.
- Think a last time about renaming tables or attributes.
- If two tables have the same key, consider merging them ("consider" does not mean to do it always!).
- Check the generated tables for a relational normal form (e.g. 3NF, BCNF, 4NF) (see next chapter).