

2. The Entity-Relationship Model

This section's goal:

After completing this chapter, you should be able to

- explain the **three phases of database design**,
Why are multiple phases useful?
- evaluate the significance of the **Entity-Relationship Model** (ER model) for DB design,
- enumerate the basic constructs of the ER model,
- develop **ER diagrams** (schemas in the ER model) for a given application,
- **translate** ER models into equivalent (as far as possible) relational models.

Database design (1)

- Overall goal of DBMS usage: Efficiently develop programs to support given real-world tasks.
- These programs need to **store data persistently**.
- To develop these programs, apply proven methods of **software engineering**—specialized to support **data-intensive** programs.

Definition (Database Design)

Database Design is the process of developing a **database schema** for a given application.

DB design 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 develop once the structure of the data to be manipulated has been specified.



Data, however, is an **independent resource**:

- Typically, additional programs will be developed later based on the collected data.
- Also, **ad-hoc queries** will be posed against the DB.

Database design (3)

During DB design, a **formal model of the relevant aspects of the real world** (“mini world”, “domain of discourse”) must be built.

Once the DB is up and running, questions will be posed against the DB. Knowledge of these questions beforehand is important input to the DB design process and helps to identify the relevant parts of the real world.

In some sense, the real world is the **measure of correctness** for the DB schema: the possible DB states should correspond to the states of the real world.

Database design (4)

DB design is not easy for a variety of reasons:

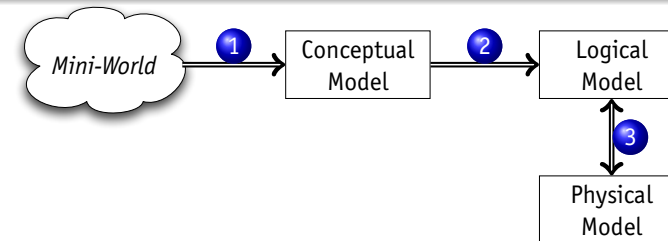
- **Expertise:** The designer must become an expert for the application domain or needs to talk to such experts. Depending on the domain, these experts might *not* be used to give a complete and formal account of their field.
- **Flexibility:** The real world typically permits exceptions and corner cases. Does the DB schema need to reflect these?
- **Size:** Database schemas may become huge (in terms of number of relations, attributes, constraints).

Due to this complexity, DB design is a **multi-step** process.

Database design (5)

Definition (Three Phases of DB Design)

- 1 **Conceptual Database Design.** Produces the initial model of the mini world in a **conceptual data model** (e.g., in the ER model).
- 2 **Logical Database Design.** Transforms the conceptual schema into the data model supported by the DBMS (e.g., the relational model).
- 3 **Physical Database Design.** Design indexes, table distribution, buffer size, etc., to maximize performance of the final system.



Database design (6)

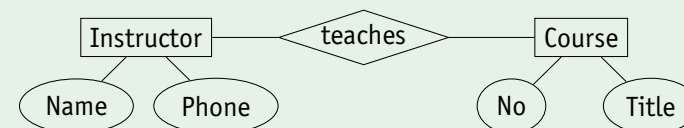
Why multiple design phases?

- Partition the problem, attack one sub-problem after the other.
For example, during conceptual design, there is no need to worry about performance aspects or limitations of the specific SQL dialect of the RDBMSs.
- DBMS features do not influence the conceptual design and only partially influence the logical design.
Thus, the conceptual design work is not invalidated, if a different DBMS is used later on.

Example (1)

ER schemas are typically shown in graphical notation:

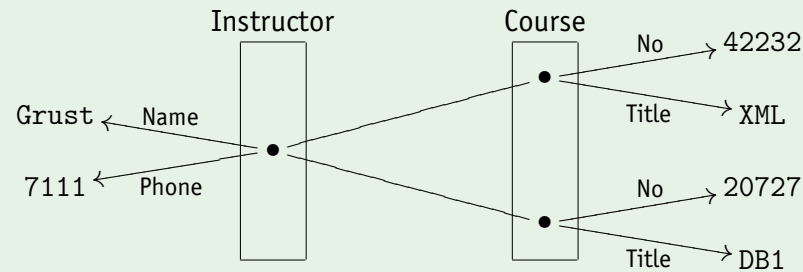
Example ("Classical" ER graphical notation:)



- This mini world talks about **instructors** and **courses**.
- Instructors **teach** courses.
- Instructors have a **name** and a **phone number**.
- Courses are **enumerated** and have **titles**.

Example (2)

Example (A possible state of this mini world)



The ER model (1)

The **Entity-Relationship Model** is often referred to as a **semantic data model**, because it more closely resembles real world scenarios than, *e.g.*, the relational model.

- In the ER model, we model the concept of “Instructors.” In the relational model we deal with names and phone numbers.
- In the ER model, there is a distinction between **entities** (objects) and **relationships** between such entities. In the relational model, both concepts are represented by relations.

The ER model (2)

- Proposed by Peter Chen (1976), today at Louisiana State University (<http://bit.csc.lsu.edu/~chen/chen.html>).
- The ER model comes with a **graphical notation** which helps to establish quick overviews of database application schemas.
Such ER diagrams are also a great way to communicate schemas to non-expert/future DB users.
- There are *no* “ER Model DBMS”.
Instead, we will describe a translation of ER model concepts into the relational model.

Basic ER model concepts (1)

Entities:

- An **object** in the mini world about which information is to be stored. Examples: *persons, books, courses*.
Note: entities do not have to correspond to objects of physical existence. Entities may also represent conceptual objects like, e.g., vacations.
- The mini world that has to be modelled can contain only a **finite number of objects**.
- It must be possible to distinguish entities from each other, *i.e.*, objects must have some **identity**.
Examples: entity book identified by ISBN number, entity vacations identified by travel agency booking number.

Basic ER model concepts (2)

Attribute:

- A **property** or feature of an entity (or relationship, see below).
Example: the title of this course entity is "Databases."
- The **value** of an attribute is an element of a data type like string, integer, date.
These values have a **printable representation** (which entities have not).

Basic ER model concepts (3)

Relationship:

- A **relation**—not in the strict relational model sense—**between pairs of entities** (a binary relationship).
Example: Grust (an entity) teaches (a relationship) the course "Foundations of Databases" (an entity).
- "Relationship" is often used as an abbreviation for "Relationship type" (see below).

Basic ER model concepts (4)

- **Entity Type:**
 - A set of **similar entities** (similar with respect to the information which has to be stored about them), *i.e.*, entities which have the same attributes.
Example: All faculty members of an institute.
- **Relationship Type:**
 - A set of **similar relationships**, *i.e.*, relationships which have the same attributes.
Example: X teaches course Y.

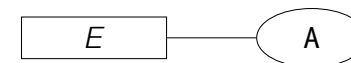
ER diagrams (1)

In the literature, you will find a vast variety of graphical notations for ER diagrams. Here, we use the "classical" boxes-and-diamonds symbols.

- **Entity type E :**



- **Attribute A of entity type E :**

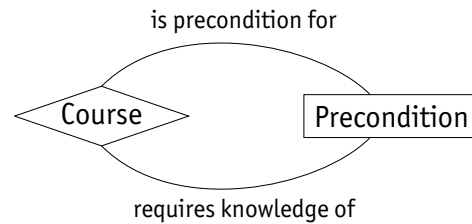


- **Relationship R between entity types E_1 and E_2 :**



ER diagrams (2)

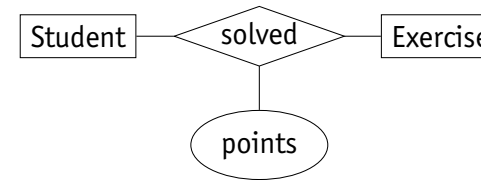
- Relationships may also exist between **entities of the same type** (sometimes misleadingly called “recursive relationships”):



- In this case, **role names** have to be attached to the connecting edges.
In the ER model, role names may be attached to any kind of relationship for documentation purposes.

ER diagrams (3)

Relationships may have attributes, too:



- This models the fact that a number of *points* is stored for every pair of a student X and an exercise Y such that X submitted a solution for Y .

Graphical syntax

- An ER diagram contains
 - boxes, diamonds, ovals, plus interconnecting lines.
- Boxes, diamonds, and ovals are each **labelled** by a string.
 - Box labels are unique in the entire diagram.
 - Oval labels are unique for a single box or diamond.
 - Diamond labels are unique for a pair of connected boxes.
- Interconnecting lines are only allowed between
 - box—diamond, box—oval, diamond—oval.
- A diamond has exactly two connecting lines to boxes. There may be any number of connections to ovals.²
- An oval has exactly one connecting line.

²But see N -ary relationships below.

ER example

Modelling a mini world: Define an ER diagram

- Information about researchers in the database field is to be stored.
- For each researcher, his/her last name, first name, e-mail address, and homepage URI are relevant.
- Researchers are affiliated with universities and assume a certain position (*e.g.*, professor, lecturer).
- Relevant university information are the name, homepage URI, and country.
- Researchers publish articles (of a given title) in journals.

ER schemas: Formalizing the semantics (1)

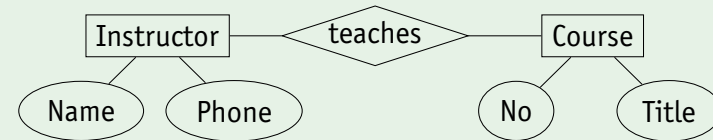
Definition (ER: Database State)

A **DB state** I interprets the symbols in the ER schema by defining

- a finite set $I(E)$ for every entity type E ,
- a mapping $I(A):I(E) \rightarrow \text{val}(D)$ for every attribute A of an entity type E , where D is the data type of A and $\text{val}(D)$ is the domain 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 $I(A):I(R) \rightarrow \text{val}(D)$ for every attribute A of relationship R (D is the data type of A).

ER schemas: Formalizing the semantics (2)

Example state corresponding to ER schema

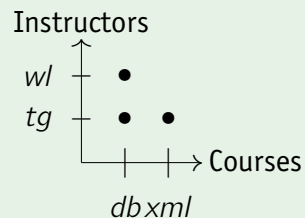


- $I(\text{Instructor}) = \{tg, wl\}$
- $I(\text{Name}) = f_1$ with $f_1(tg) = \text{'Grust'}$, $f_1(wl) = \text{'Lex'}$
- $I(\text{Phone}) = f_2$ with $f_2(tg) = 7111$, $f_2(wl) = 7132$
- $I(\text{Course}) = \{db, xml\}$
- $I(\text{No}) = g_1$ with $g_1(db) = 20727$, $g_1(xml) = 42232$
- $I(\text{Title}) = g_2$ with $g_2(db) = \text{'DB1'}$, $g_2(xml) = \text{'XML'}$

ER schemas: Formalizing the semantics (3)

Example state corresponding to ER schema (cont.)

- $I(\text{teaches}) = \{(tg, db), (tg, xml), (wl, db)\}$

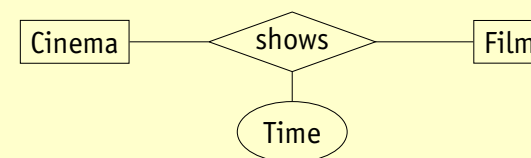


- **Note:** A relationship is interpreted by a set of **entity pairs** and thus is unique for two given entities.

ER schemas: Formalizing the semantics (4)

Consequences of the ER semantics

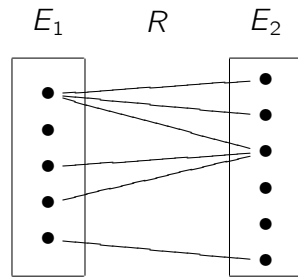
Consider the ER diagram below:



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

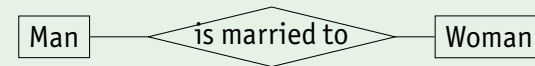
Cardinalities (1)

General ER relationship:



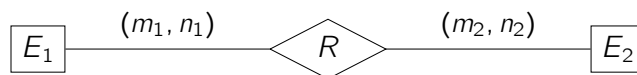
Cardinalities (2)

- In general, there is **no restriction** on how often an entity participates in an relationship R .
 - An entity can be connected to one entity of the other type, to more than one, or it can have no R -partner at all.
- However, specific **application semantics** dictate to how many E_2 entities an E_1 entity can be related:



Cardinalities (3)

The ER model introduces the **(min, max) notation** to specify an **interval** of possible participations in an relationship:



- An entity of type E_1 may be related to at least m_1 and at most n_1 entities of type E_2 .
- Likewise, m_2 is the minimum number and n_2 is the maximum number of E_1 entities to which an E_2 entity is 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 $e_2 \in I(E_2)$:

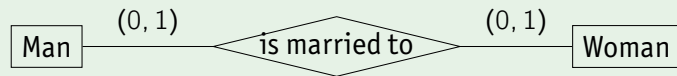
$$m_2 \leq |\{e_1 \in I(E_1) \mid (e_1, e_2) \in I(R)\}| \leq n_2$$

Extensions:

- “*” may be used as maximum, if there is **no limit**.
- (0, *) means no restriction at all (general relationship).

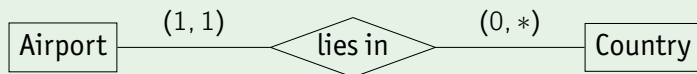
Cardinalities (5)

Example (Marriage)



"A man can be married to at most one woman and vice versa."

Example (Airport Locations)

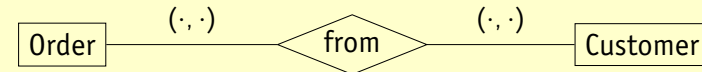


"An airport lies in exactly one country. A country may have arbitrarily many airports (and maybe none at all)."

Cardinalities (6)

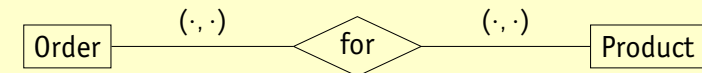
Derive cardinalities from verbal specifications

"Besides normal customers, the database may contain customers who have not yet ordered anything."



Derive cardinalities from verbal specifications

"An order can contain several products."



Selecting cardinalities

- Sometimes a sketch of a valid database state may help in selecting the right cardinalities, *e.g.*, for the state sketched on slide 71, a viable cardinality for E_1-R may be (0, 3).
- Application knowledge might lead to **weaker restrictions**, in this example (0, 5) or (0, *).
A cardinality (a, b) is **weaker** than (c, d) if $a \leq c$ and $d \leq b$.
- In real applications, the cardinalities (0, 1), (1, 1), and (0, *) are the most common and especially **easy to enforce in the relational model**.

Common cases (1)

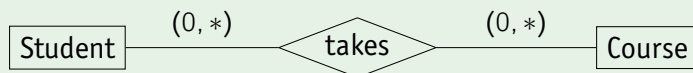
- Normally, the **minimum cardinality** will be 0 or 1, and the **maximum cardinality** will be 1 or *.
 - Thus, only the (0, 1), (1, 1), (0, *), (1, *) cardinalities are common in practice.
- To understand a relationship, one must know the cardinality specifications on **both sides**.
- The **maximum cardinalities** on each side 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 are 0 or 1):

Many-to-many relationship



- This is the most general/least restrictive case of a relationship.
- When translated into the relational model, the representation of many-to-many relationships requires an extra table.

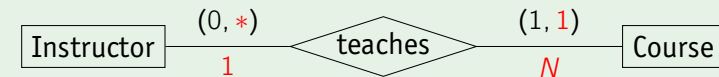
Common cases (3)

One-to-many relationships:

- Maximum cardinality 1 on the "many" side and * on the "one" side:³

One-to-many relationship

"One instructor teaches many courses, but each course is run by exactly one instructor."



- One-to-many relationships do *not* require an extra table in an equivalent representation in the relational model.

³Notice the antisymmetry!

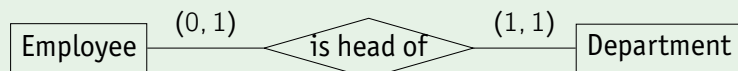
Common cases (4)

One-to-one relationships:

- Maximum cardinality 1 on both sides:

One-to-one relationship

"Each department has exactly one department head, some employees are the head of one department."



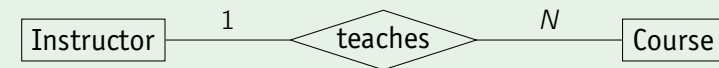
- Note how **mandatory** or **optional** participation in a relationship determines the **minimum cardinalities**.

Cardinalities: Alternative notations (I)

Widespread variants of notations for cardinalities include the following:

- Leave participation (mandatory/optional) unspecified: Cardinalities are either **many-to-many** ($N:M$), **one-to-many** ($1:N$), or **one-to-one** ($1:1$).

One-to-many relationship

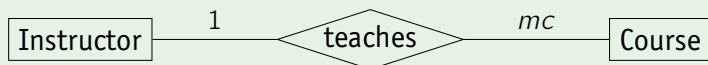


Cardinalities: Alternative notations (2)

- 2 Indicate participation (mandatory/optional) by the use of “c” (for *conditional*) in place of “1” or “mc” in place of “N”: Cardinalities can thus be **many-to-many** ($m[c]:n[c]$), **one-to-many** ($1|c:m[c]$), or **one-to-one** ($1|c:1|c$).

One-to-many relationship

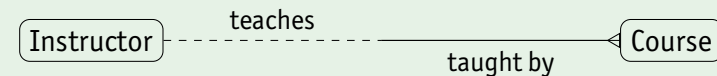
“Instructors may or may not teach course(s), each course has exactly one instructor.”



Cardinalities: Alternative notations (3)

- 3 “Crow foot” notation: Sometimes found in software supporting visual ER diagram development (e.g., in Oracle Designer™):

One-to-many relationship

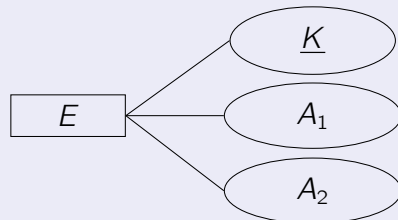


- The “crow foot” indicates the “many” side.
- Dashed lines indicate optional participation.
- Role names are given for both sides of the relationship.
- 4 Many more can be found! For instance, UML class diagrams use yet another notation.

Keys (1)

Definition (ER Key)

A **key** K of an entity type E^4 is an attribute of E which **uniquely identifies** the entities of this type. No two different entities share the same value for the key attribute, i.e., $I(K)$ is **injective** for all DB states I . Composite keys are allowed.




⁴Only *entity* types can have key attributes.

Keys (2)

Importance of keys:

- The ER model does **not require** the declaration of a key for each entity type E since entities already have a distinguished “object identity”, namely the elements of $I(E)$.

*But see **weak entities** introduced shortly.*

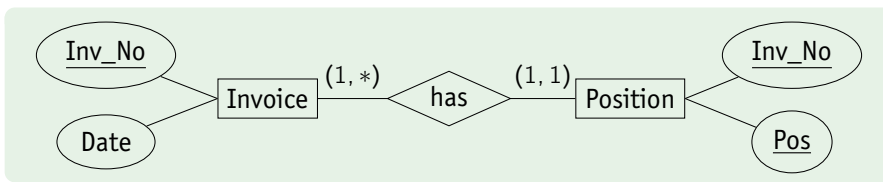
- The **translation of ER schemas into relational schemas** requires the declaration of ER keys, though. 
- If there is no natural key, add **artificial identifiers** (e.g., integers, remember attributes EMPNO, DEPTNO from above) which then represent the elements of $I(E)$.

Weak Entities (1)

In many schemas, some entities describe a kind of **detail** that **cannot exist** without a **master** (or owner) **entity**.

In such a case,

- 1 there is a **relationship with cardinality (1, 1) on the detail entity side**, and in addition
- 2 the **key of the master is inherited and becomes part of the key of the detail entity**.



Weak Entities (2)

- Without a specific ER construct for this case, we would require the following additional constraint:
 - If two entities are in “has” relationship,
 - then their attribute “Inv_No” are required to have identical values.

For example, invoice #12 cannot have position 2 in invoice #42 as detail.
- Such constraints occur, if **an entity does not have a key by itself, but it is only unique in the context of some other (master) entity**.

Weak Entities (3)

Note: In weak entities, keys are **always composite**.

- Examples:
 - A classroom is identified by a building *and* a room number.
 - A section in a book is identified by a chapter *and* a section title.
 - A web page URI is composed of a web server DNS address *and* a path on that server.

There is also an **existence dependency**.

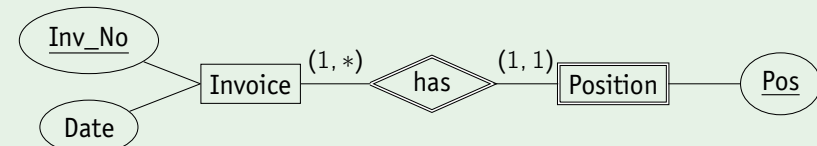
- If the building is pulled down, the classrooms automatically disappear.
- If the web server is shut down, all URIs on that server cease to function.

Weak Entities (4)

In the ER model, such scenarios are modelled via **weak entities**⁵.

In ER diagrams, weak entities and their identifying relationships are indicated by **double lines**:

Weak Entities



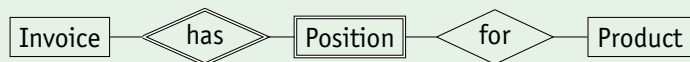
For the weak entity, the **inherited part of the key is not shown**.

⁵Non-weak entities are also called **strong entities**.

Weak Entities (5)

- Usage of weak entities is required only if the key of one entity **contains** the key of a related entity.
A weak entity adds further key attributes to the master key.
- Weak entities are normal entities except that their key is constructed in a special way.

Weak entities can take part in other relationships



- Weak entities may themselves be masters of other weak entities (building an entire hierarchy of dependencies).

Weak Entities (6)

Modelling with weak entities

Model a set of online quizzes (multiple choice tests).

- Each quiz is identified by a title, each question within a quiz is numbered, and each possible answer to a given question is referenced by a letter.

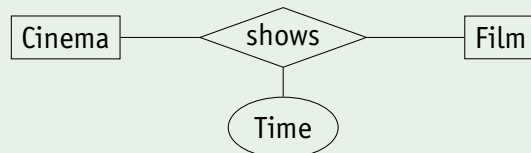
*For each question and answer, the associated text is stored.
Answers are classified into correct and incorrect ones.*

- What is the complete key for each of the occurring entity types?

Relationships connecting more than 2 entities

Remember our example from above:

Example (Cinema timetables)



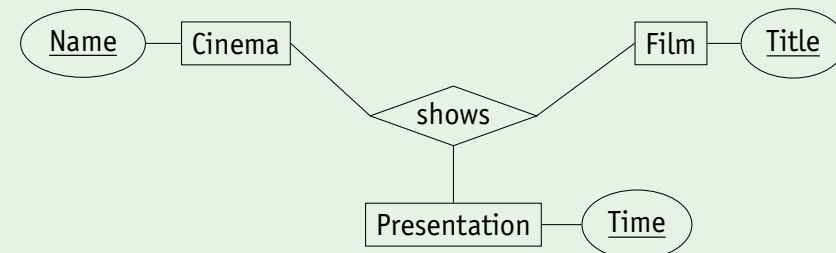
If a cinema shows the same movie more than once, we're in trouble. There are basically two options:

- Allow the use of n -ary relationships, connecting $n \geq 2$ entity types.
- (Disallowing those:) Introduce additional entity types.

N-ary relationships

We allow relationships that "connect" more than 2 entity types. In the example, a *ternary* relationship will do:

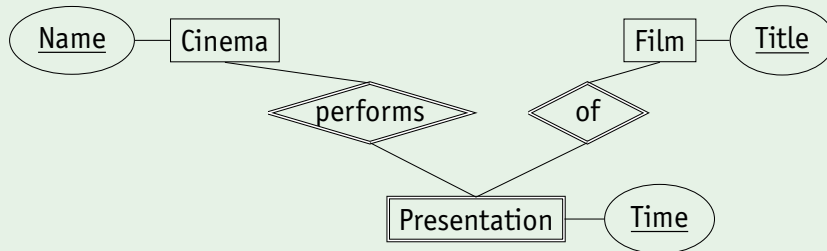
Cinema timetables with ternary relationship



Association Entities

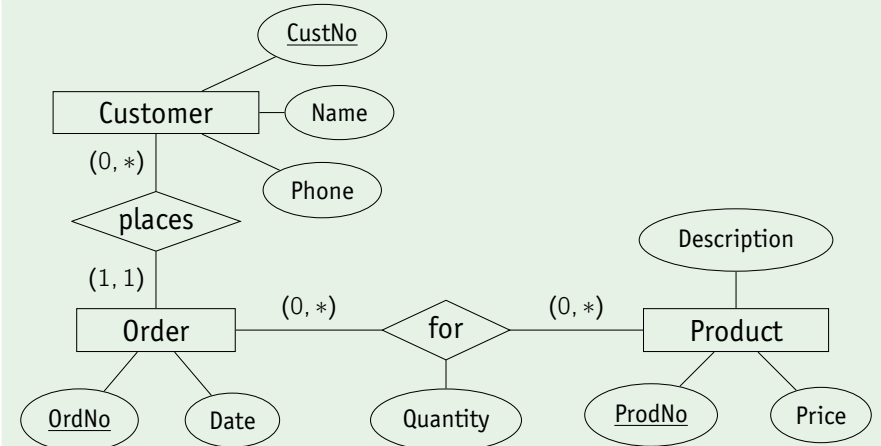
Use an **association entity**: turn relationship into weak entity with multiple owners. Inherits keys of all owners.

Cinema timetables with association entity



Translation into the relational model

ER diagram example



Step 1: Entities (1)

Transforming an ER entity E :

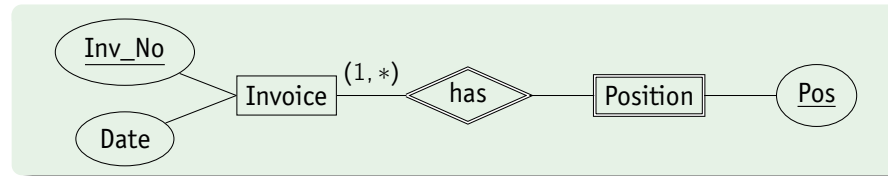
- 1 Create a **table for each entity**.
The table name is E (conventionally: $E + 's'$).
- 2 The **columns** of this table are the **attributes of the entity type**.
- 3 The **primary key of the table** is the **primary key of the entity type**.
*If E 's key is composite, so will be the relational key. If E has no key, add an **artificial key** to the table.*

Step 1: Entities (2)

Customers			Orders	
<u>CustNo</u>	Name	Phone	<u>OrdNo</u>	Date
10	Jones	624-9404	200	2/15/04
11	Smith		201	2/16/04

Products		
<u>ProdNo</u>	Description	Price
1	Apple	0.50
2	Kiwi	0.25
3	Orange	0.60

Step 1 B: Weak entities



When a **weak entity** is translated, the **key attributes of the owner entity** are added as a **key and foreign key**:

Position (Pos, Inv_No → Invoice, ...)

- This automatically implements the relationship.
- In SQL, it makes sense to specify ON DELETE CASCADE for the foreign key: if an invoice is deleted, all its positions will be removed from the DB state, too.

Step 2: One-to-many relationships (1)

Transforming a relationship R :

- 1 If R has **maximum cardinality 1 on one side**, R is **one-to-many**⁶.
*Example: Customer-(0, *)-places-(1, 1)-Order, "one customer places many orders."*
- 2 In this case, add **key of "one" side as a column to the "many" table** to implement R .
- 3 This column will be a **foreign key referencing a row in the table representing the related entity**.

⁶If R has maximum cardinality 1 on both sides, it is actually one-to-one, see below.

Step 2: One-to-many relationships (2)

Orders (OrdNo, Date, CustNo → Customers)

Orders		
<u>OrdNo</u>	Date	CustNo
200	2/15/04	11
201	2/16/04	11

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

Convention: use relationship and role to name foreign key column:

Orders (OrdNo, Date, placed_by → Customers)

Step 2: One-to-many relationships (3)

Transforming a one-to-many relationship R (cont'd):

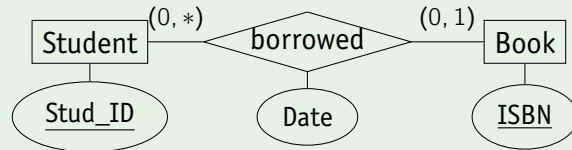
- If the **minimum cardinality is 1** on the "many" side (see example), null values are **not allowed** in the foreign key column (column placed_by in example).

If the **minimum cardinality is 0**, null values **are allowed** in the foreign key column.

The foreign key is null for those entities that do not participate in R at all.

Step 2: Relationship attributes

To transform **one-to-many relationship attribute(s)**, e.g.



- store the relationship attribute(s) together with the reference to the related entity, e.g.

Example

Books (ISBN, ..., borrowed_by → Students, Date)

Step 2: One-to-many relationships: Variant

One-to-many relationships R with cardinality **(0,1)** can be translated into a table of their own:

borrowed_by (ISBN→Books, Stud_ID→Students, Date)

- The extra table holds the key values of the related entities plus the relationship attributes.
- The key attributes of the side with the (0,1) cardinality become the key of this relation.

"Each book can be borrowed only once at the same time."

This does not model R correctly if the cardinality is (1,1)

Why?

Step 3: Many-to-many relationships (1)

Transforming a many-to-many relationship R :

- 1 If R has maximum cardinality $*$ on **both sides**, R is **many-to-many**.
Example: Order-(1,)-for-(0,*)-Product, "an order contains many products, a product may be part of many orders."*
- 2 R becomes its **own table**.
- 3 The columns of this table are the **keys of both participating entity types**.
- 4 These columns act as **foreign keys** referencing the entities and, at the same time, together form a **composite key** for the extra table.

Step 3: Many-to-many relationships (2)

Transforming a many-to-many-relationship R (cont'd):

- 5 Relationship attributes are added as columns to the table representing R .

for (OrdNo→Orders, ProdNo→Products, Quantity)

Composite key?

Is it really necessary that both entity keys (here: OrdNo, ProdNo) form a composite key for the relationship table?

Step 3: Many-to-many relationships (3)

for (OrdNo → Orders, ProdNo → Products, Quantity)

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

Orders			Products		
OrdNo	Date	CustNo	ProdNo	Description	Price
200	2/15/04	11	1	Apple	0.50
201	2/16/04	11	2	Kiwi	0.25
			3	Orange	0.60

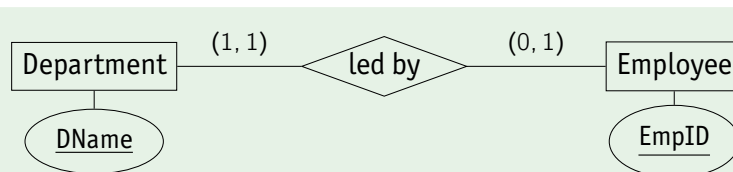
Step 3: Many-to-many relationships (4)

Transforming a many-to-many relationship R (cont'd):

- **Note:** Minimum cardinalities other than 0 for R **cannot be enforced** by the relational model per se, *i.e.*, in terms of key constraints.
 $Order-(0, *)-for-(1, *)-Product \Leftrightarrow$ "Every product occurs in at least one order."
- If this is important to guarantee database consistency (valid DB state), this constraint needs to be **checked by the application program** or a **general RDBMS constraint mechanism**.

Step 4: One-to-one relationships (1)

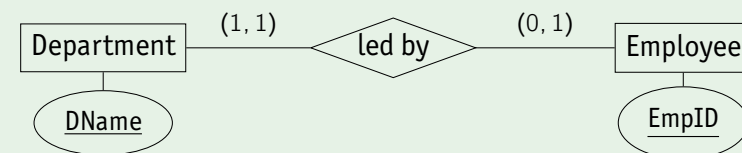
Transforming a one-to-one relationship R :



- 1 If R has maximum cardinality 1 on **both sides**, R is **one-to-one**.
- 2 We can essentially transform as if R were one-to-many, but additional key constraints are generated.

Step 4: One-to-one relationships (2)

To which entity table shall we add the `led_by` attribute to represent the relationship?

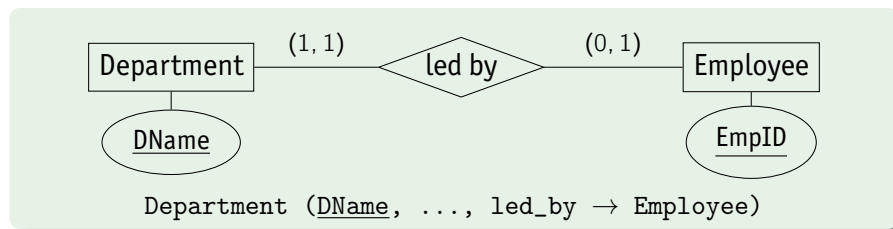


- Since we have Department--(1, 1)--led_by ("every department is led by exactly one employee"), it makes sense to host the relationship in the Department table:

Department (DName, ..., led_by → Employee)

- We may declare the foreign key `led_by` as NOT NULL. (This is not possible if `led_by` is hosted in the Employee table.)

Step 4: One-to-one relationships (3)



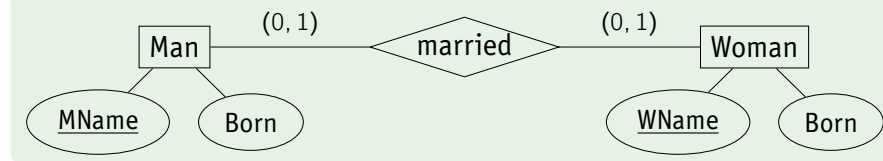
- **Note:** led_by now also is a key for the Department table.

led_by is a key for table Department

Why is this the case in the example?

- This key constraint enforces the maximum cardinality of 1 (on the Employee side).

Step 4: One-to-one relationships (4)



Two variants:

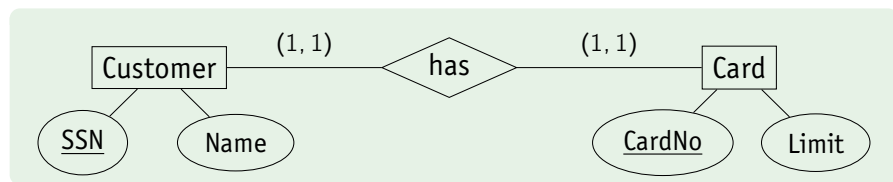
- 1 Any of the two (*not both!*) entity tables may host the married foreign key (null values allowed).
- 2 Translate the relationship into a table of its own:

married (MName → Man, WName → Woman)

A one-to-one relationship in an extra table

What would be the correct key(s) for table married?

Step 4: One-to-one relationships (5)



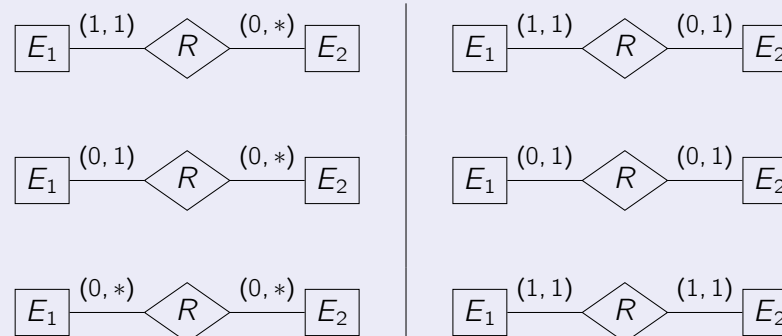
In order to **enforce the minimum cardinality 1 on both sides, the entity tables need to be merged:**

CustomerCard (SSN, Name, CardNo, CreditLimit)

- No null values are allowed.
- Both, SSN and CardNo, are candidate keys of this table. One is selected as primary key, the other is an alternative key.

Limitations (1)

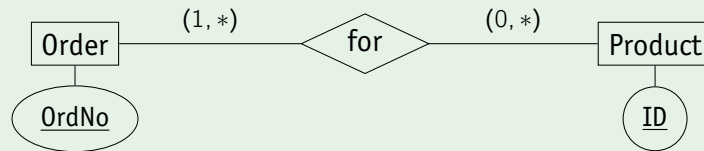
The following ER relationship cardinalities can be faithfully represented in the relational model:



For all other cardinalities, the constraint mechanisms of the relational model will not suffice.

Limitations (2)

Example



Relational (foreign) key constraints *cannot* enforce the minimum cardinality 1 (“Every purchase order includes at least one product item.”)

⇒ Apply the translation method for $(0, *)$ cardinalities and enforce the constraint in the application (or via a separate constraint mechanism in the RDBMS, e.g., a **trigger** whenever a tuple in table Order is inserted).

A note on ER attributes

Some ER variants permit the use of non-atomic attributes. For example, attributes may sometimes be

- composed of several components (like a record/struct in a programming language),
... *address = (street, no, zip, city, state)*
- array- or set-valued,
... *given_names = name**
- or even combinations thereof.

In such cases, due to the *first normal form* (1NF) restriction in an RDBMS, ER attributes need to be mapped in a non-trivial manner, e.g.

- onto several attributes in case of record-composition,
- onto separate tables in case of multi-valued attributes.

Extended ER modelling

The basic ER model discussed so far has been extended in a variety of directions. Most notably, **generalization/specialization** hierarchies have been introduced.

Generalization ... denotes the process to “factor out” common characteristics (e.g., attributes, constraints) of several entity types into a more general, common **supertype entity**.

Specialization ... denotes the opposite process: deriving new, specialized **entity subtypes** from a given type, possibly adding specific characteristics (such as, e.g., more attributes or more restrictive constraints).

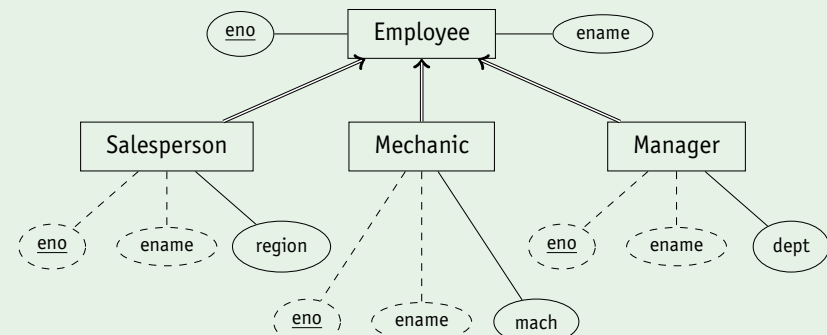
Inheritance ... means the fact that (no matter how we ended up with the hierarchy) entity subtypes fulfill all restrictions (presence of attributes, their domains, constraints) that are defined for the supertype. They can, however, strengthen them.

Generalization vs. Specialization

Generalization: when we start from the subtypes (going up ↑).

Specialization: when we start from the supertypes (going down ↓).

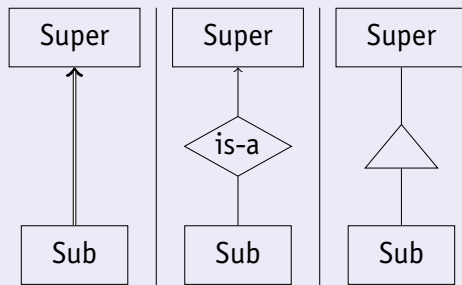
Generalization vs. specialization: different types of employees



Notation

... again, a number of different notations are in use. Some of them are given below:

Graphical syntax for generalization hierarchies ("IS-A"-relationships)



Additional constraints

The following additional constraints can—sometimes—be specified with generalizations/specializations:⁷

disjoint ... If S_1 and S_2 are subtypes of the same supertype E , no E can be an S_1 and an S_2 at the same time.

No Animal can be a Fish and a Mamal.

covering ... If S_1 and S_2 are subtypes of the same supertype E , each E must either be an S_1 or an S_2 (or both).

All Persons are either Men or Women, there are no "others".

derived ... E has an attribute A , whose value determines whether it is an S_1 or an S_2 .

Employees have a distinguishing Job_title attribute.

⁷All of these are not restricted to *two* subtypes!

More on inheritance

- Inheritance means that *all specifications* are inherited from a supertype E to all its subtypes. This includes, for example:
 - the attributes and key(s) of E ,
 - all relationships E is involved in and their cardinalities,
 - any additional constraints that may be specified for E .

Each of these can be *refined* in the subtype.

Refining specifications ...

Give examples for such refinements!

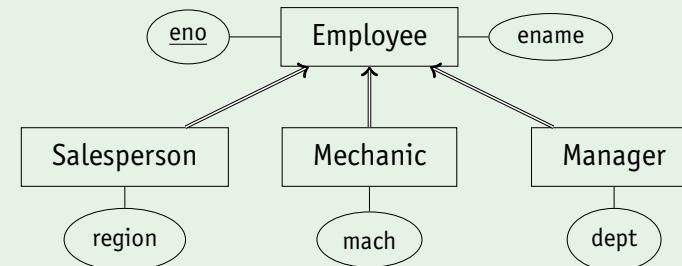
What would **not** be a refinement and, hence, not be permitted?

- Inheritance can be restricted to *single inheritance* (as in Java), *i.e.*, each type can have at most one supertype, or
- (more generally), *multiple inheritance* may be permitted, such that a subtype can have multiple generalizations.

Mapping extended ER models to relational schemas

The relational model provides no direct representation for generalizations. Thus, we need to use auxiliary constructions.

Example (Generalization hierarchy)



Relational modelling of generalizations (1)

Option 1: Each (super, sub) entity type is mapped to a separate relation.

Example (Separate relations)

Employee	(<u>eno</u> , ename)	
Salesperson	(<u>eno</u> , region)	[eno → Employee]
Mechanic	(<u>eno</u> , machine)	[eno → Employee]
Manager	(<u>eno</u> , dept)	[eno → Employee]

Notice:

- Each table contains only those attributes that are specific for this type (no inherited attributes).
- The key is inherited down the hierarchy (also as a foreign key).
- Access to inherited attributes within a subtype requires joins.
- No redundant storage (except for the keys) is introduced.

Relational modelling of generalizations (2)

Option 2: Use only one relation for all types.

Example (One single relation)

Employees (eno, ename, region, machine, dept)

Notice:

- Lots of NULL values are introduced.
- A discriminating attribute (such as Job_title, in our example) might be introduced to support testing the kind of entity at hand.
- No joins are necessary to access all attributes.
- Hardly viable in case of multiple inheritance (redundancy)!

Relational modelling of generalizations (3)

Option 3: Use independent relations for subtypes.

Example (Independent subtype relations)

Salesperson	(<u>eno</u> , ename, region)
Mechanic	(<u>eno</u> , ename, machine)
Manager	(<u>eno</u> , ename, dept)

Notice:

- Concept of generalization is lost.
- Redundancy in schema (constraints)!
- Redundancy in tuples in case of non-disjoint specializations!
- Not possible with non-covering specializations (or supertype relation needed in addition).
- No joins are necessary to access all attributes.

Relational modelling of generalizations (4)

More options ... and how to select?

- Parts of the options shown can be combined.
- In a larger generalization hierarchy, choices are not completely independent.
- Guidelines for the selection may be obtained by considering, *e.g.*
 - ease of use (how difficult to express are typical queries?),
 - additional integrity constraints (which constraints can be expressed at all/easily?),
 - the rest of the schema (which subtypes are involved in which relationships?),
 - performance criteria (how efficient are the most frequent operations?).

3. Relational Normal Forms

This section's goal:

After completing this chapter, you should be able to

- work with **functional dependencies** (FDs),
define them, detect them in DB schemas, decide implication, determine keys
- explain insert, update, and delete **anomalies**,
- explain **BCNF**, test a given relation for BCNF, and transform a relation into BCNF,
- detect and correct violations of **4NF**,
- detect **normal form violations** on the level of ER,
- decide if and how to **denormalize** a DB schema.