Part 5: ER-Model III

Elmasri/Navathe:

- Ch. 3, "Data Modeling Using the Entity-Relationship Model"
- Ch. 4, "Enhanced Entity-Relationship and Object Modeling"

I have also used the following sources:

- 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"
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German), Teubner, 1997.
- Kemper/Eickler: Datenbanksysteme (in German), Ch. 2, Oldenbourg, 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

- You should be able to explain the advanced ER-constructs introduced in this chapter,

  i.e. weak entities, optional/multivalued/structured attributes, and specialization (subclasses),
- to use them correctly in designing ER-diagrams for given applications,
- and to compare and critize ER-schemas containing them.
- In addition, you should know some sources of information (input) for database design,
- and have some impression of the functionality of design tools.

  These topics will be treated in greater depth in INFSCI 2711.

Overview

Extended ER-Models (1)

- Many extensions to the basic ER-model have been proposed.
- The goals are to capture more meaning from the real world and to simplify the conceptual design.
- Especially, we would like to avoid additional constraints whenever possible.

  If we can express a situation formally with constructs of the model, this is better than adding constraints in natural language. E.g. we can then give a formal translation into the relational model.
- Also, additional constructs may help explain the relation to the real world better (like a comment).

Extended ER-Models (2)

- Adding more constructs to the ER-model gives Extended ER (EER)-models.
- There are many different proposals for such models.

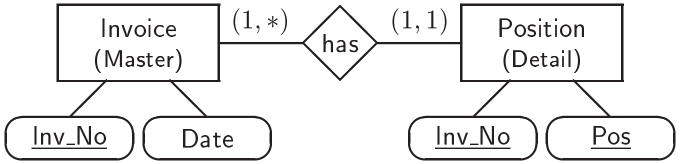
  Therefore, the notation for the extended constructs varies quite a bit among different authors.
- Which extensions you can use depends on policies of your company and the available CASE tools.
- Most extensions make the translation into the relational model harder, but when they simplify the difficult conceptual design, this is worth problems during logical design.

Overview

1. General Remarks on Extended ER-Models
2. Advanced ER-Constructs I: Weak Entities
3. Advanced ER-Constructs II: Optional, Multivalued, Structured Attributes
4. Advanced ER-Constructs III: Specialization
5. Misc. Remarks on DB Design (Under Construction)

Weak Entities (1)

- Sometimes, an entity describes a kind of "detail" and cannot exist without a "master" or "owner" entity.
- So 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:



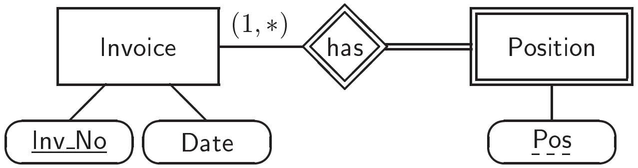
Weak Entities (2)

- Without a specific construct for this situation, we would need the constraint that
  - if two entities are connected via the relationship "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.
- So in these cases we always have 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).

Weak Entities (3)

- Weak entities were introduced for such situation where an entity needs to borrow key attributes from a related entity.
- Weak entities are marked in ER-diagrams by doubling their box, the connecting line, and the relationship diamond:



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

Weak Entities (4)

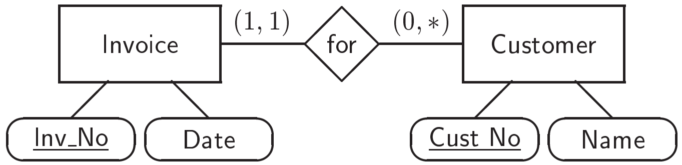
- So the real key of the weak entity consists of the key attribute of the "owner" entity type (automatically inherited, not explicitly shown) plus the dashed-underlined partial key.
- Another way to see this is that here the relationship contributes to the identification of the entity, whereas usually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

  Some authors use bullets to mark key attributes and thus put here a bullet at the connection to the relationship.
- The cardinality $(1, 1)$ is automatically 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.

Weak Entities (5)

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



- A weak entity is only needed 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 ($\rightarrow$ below).

Weak Entities (6)

- Sometimes the master/owner entity is called "parent entity", and the dependent weak entity is called "child entity".
- Entities with their own key are called regular/strong entities.
- Weak entities can be owner entities for other weak entities.

  So you can have an entire hierarchy of parent-child relationships (e.g. "grandchildren").
- Cycles are forbidden: It is impossible that A is a weak entity with owner B, and B is a weak entity with owner A.
- Exercise: Model a set of online quizzes. Each quiz is identified by a title, each question within a quizz by a number, and each answer to a given question by a letter.

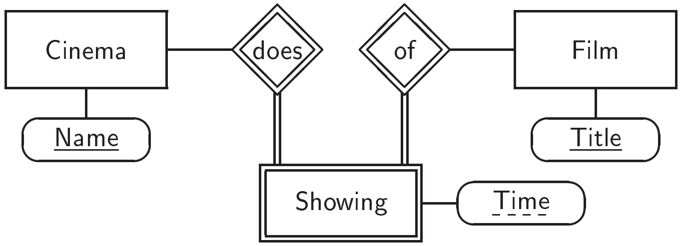
  What is the complete key of each of the three entity types?

Association Entities (1)

- Sometimes we want to turn a relationship into an entity, e.g. because:
  - The relationship is ternary and our version of the ER-model supports only binary relationships.
  - The relationship is many-to-many, and we want only one-to-many relationships.
  - We want to define a relationship between this relationship and some entity type.
  - A relationship-attribute does not have a unique value.
- We then create a weak entity which inherits the keys of the entity types which participate in the original relationship.

Association Entities (2)

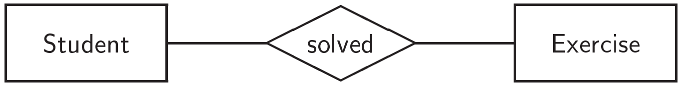
- Weak entities can have multiple parents/owners:



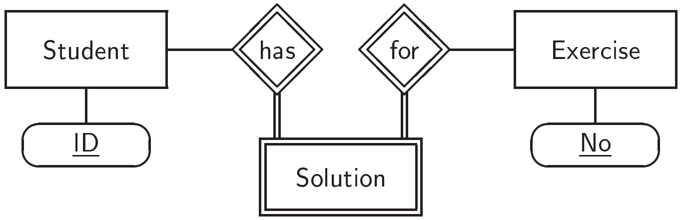
- The key of “Showing” consists then of “Name” (inherited from “Cinema”), “Title” (from “Film”), and “Time”.
- Weak entities with several parents/owners are sometimes called “association entities”. (“Weak entity” is equally ok.)

Association Entities (3)

- Relationship:



- Association Entity (completely equivalent):

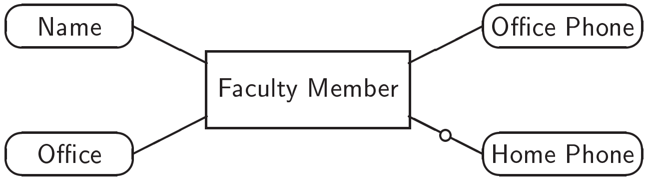


Overview

1. General Remarks on Extended ER-Models
2. Advanced ER-Constructs I: Weak Entities
3. Advanced ER-Constructs II: Optional, Multivalued, Structured Attributes
4. Advanced ER-Constructs III: Specialization
5. Misc. Remarks on DB Design (Under Construction)

Optional Attributes (1)

- Sometimes, we do not know the value of all attributes, or a value does not even exist.



- E.g. not all faculty members want their home phone number stored, newly moved ones might not even have a phone.
- So "Home Phone" is an optional attribute here.

Optional Attributes (2)

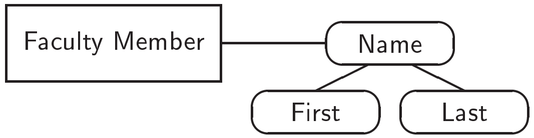
- Undefined attributes correspond to "null values" in the relational model (see below).
- The semantics of optional attributes is quite complicated, partially because a missing value can mean a variety of different things, e.g.:
  - There is a value, but it is not known.
  - There is no value.
  - The attribute makes no sense for this specific entity.
    E.g. a typing speed for a pilot.
- So try to avoid optional attributes when you can.
  But you often cannot.

Optional Attributes (3)

- In practice it quite often happens that you don't have all values at once.
  Then it is much better to allow to leave an input field blank than to force the users to enter some nonsense.
- Note that null values are not the same as the empty string: The empty string is a defined value and poses no problem.
  Allowing the empty string and a null value for the same attribute may make it difficult to distinguish between the two: What does an empty field in a screen mask/form mean?
- Key attributes cannot be optional.
  At least the primary key, which you mark by underlining, must be defined. We do not allow "anonymous entities".

Structured Attributes

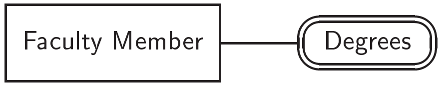
- Sometimes you want to treat values for different attributes as a whole (like a record data type).
- Also, you might want to use a "top-down" design of attributes and define their inner structure later.



- Non-structured attributes are treated as atomic.
  Their components can later only be accessed via programming.
- Structured attributes can be used in keys.
  You can either underline "Name" or "First"/"Last".

Multivalued Attributes (1)

- Sometimes, there can be several values for an attribute:



- So the attribute is in this case set-valued.
- Multivalued attributes allow the empty set (similar to null).
- This is not the same as a string with all the values concatenated (and maybe separated by some delimiter).

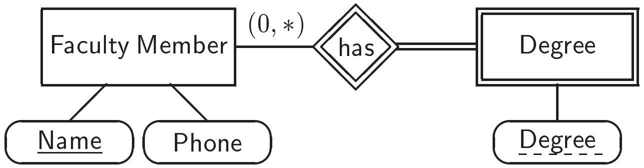
  This would be one possible encoding/translation into the relational model, but not the only one (see below).
- Multivalued attributes cannot be used as keys.

Multivalued Attributes (2)

- Multivalued attributes can always be transformed into weak entities. This is especially preferable if:
  - You want to define a relationship on the weak entity.

    Attributes cannot participate in relationships.
  - Not all attributes of the weak entity are part of the key.

    If the attribute is multivalued and structured, you cannot specify that already one field identifies each record.



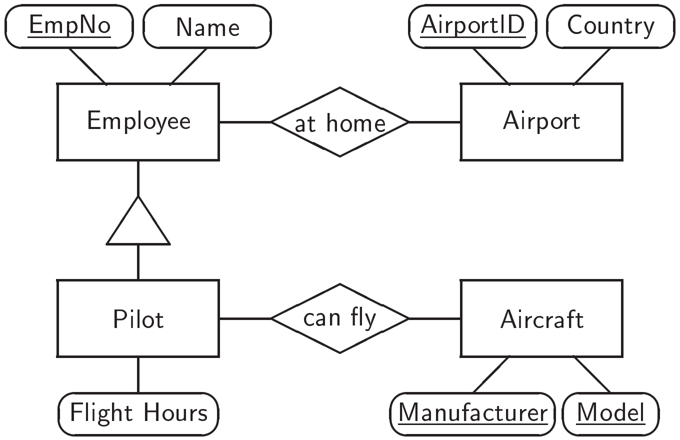
Overview

1. General Remarks on Extended ER-Models
2. Advanced ER-Constructs I: Weak Entities
3. Advanced ER-Constructs II: Optional, Multivalued, Structured Attributes
4. Advanced ER-Constructs III: Specialization
5. Misc. Remarks on DB Design (Under Construction)

Specialization (1)

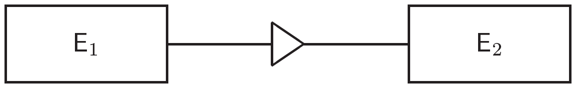
- Sometimes, two or more entity-types have attributes or relationships in common.
- Then it might be good to create a generalized entity type (“superclass”), which contains only the common characteristics, and abstracts from the differences.
- Or, some attributes or relationships apply only to a subset of the entities. Then we should consider to create a specialized entity-type (“subclass”) for this set.
- Inheritance (“is-a” relationships) have proven useful in knowledge representation and object-oriented programming languages.

Specialization (2)



Specialization (3)

- We use the following notation for specialization:



- It means that every E_1 “is a” E_2, i.e. $\mathcal{S}(E_1) \subseteq \mathcal{S}(E_2)$. If you read the triangle as an arrow, it points in the “is a” direction.
- E_1 (the subset) automatically inherits all attributes and relationships from E_2 (the superset).

  Since the set of pilots is a subset of the set of employees, all attributes defined on employees are defined on pilots in particular. If employees can participate in a relationship, this is especially true for pilots.

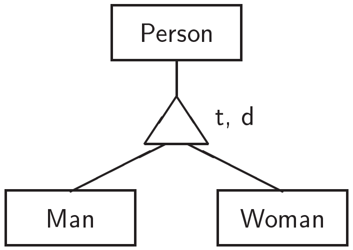
Specialization (4)

- Since attributes are inherited from the superclass, you are not allowed to define an attribute with the same name in the subclass. A similar rule applies to relationships.
- E_1 (the subclass, e.g. pilot) needs no extra key: A key of E_2 (the superclass) is automatically a key for E_1.
- E.g. if employees a uniquely identified by “EmpNo”, there especially cannot be two pilots with the same “EmpNo”.

  In exceptional cases, the inherited key might be non-minimal. E.g. if employees are identified by job and a number, the job would not be needed for pilots.
- Thus, you cannot define a primary key (underlined attributes) for subclasses. You can define alternate keys (in the text).

Specialization (5)

- An entity type can have multiple subtypes:



- It should be specified whether the subsets are disjoint (d), i.e. $\mathcal{S}(\text{Man}) \cap \mathcal{S}(\text{Woman}) = \emptyset$, or overlapping (o),
- and whether the specialization is total (t), i.e. $\mathcal{S}(\text{Man}) \cup \mathcal{S}(\text{Woman}) = \mathcal{S}(\text{Person})$, or partial (p).

Specialization (6)

- Exercise: Suppose you have an entity type for persons at the university, with the subtypes students and employees.
  - Are these subtypes disjoint?
  - What would it mean that the specialization is partial?
- Note that in most object-oriented programming languages, an object needs to have a unique type (i.e. belong to a single most-specific subclass).

  Our entity-types are not necessarily disjoint. This is sometimes a simplification. In programming languages, you would have to explicitly introduce a subclass which has as superclasses all the classes to which an object can belong (multiple inheritance).

Specialization (7)

- Subtypes can have their own subtypes.

  Students and employees are both members of a university. Students can in turn be graduate or undergraduate.
- An entity type can be specialized in different ways at the same time (orthogonal issues).

  E.g. employees can be faculty or staff members, and they can participate in a dental insurance plan or not.
- An entity type can be a subtype of multiple supertypes.

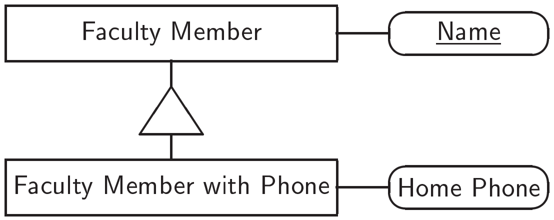
  But there must be a common supertype at the end, from which a common key is inherited. Also, each inherited attribute must be declared in a unique superclass.
  Of course, cycles in the subtype relation are excluded.

Specialization and Constraints

- Often, there are constraints related to subtypes.
- E.g., “Employee” might have an attribute “job”, and the subtype “Pilot” might then contain exactly those employees with “job=Pilot” (attribute-defined specialization).
- Such constraints can help for the translation into the relational model.
- You should also specify a constraint if the possible values for an inherited attribute are more restricted in the subtype.

Optional Attributes vs. Specialization

- Optional attributes can be represented with specialization:



- However, if various attributes can be missing, there will be a lot of such subclasses.
- So here a subclass would not be good style.
- However, if you have a group of attributes which must be either all defined or all undefined, this indicates a subclass.

Generalization / Categorization

- During specialization, we start with the superclass (e.g. Employee), and construct subclasses (e.g. Pilot).

  Conversely, we might first have the subclasses, e.g. Student and Employee, and then discover that they have common attributes, and introduce the superclass “University Member”.
- Starting from the subclasses is called generalization. The result is identical to the specialization, it is only a question in which order the different entity types are introduced.
- Some authors use the term generalization/categorization if the subclasses have keys of their own, and you want to consider their union for certain purposes (e.g. a relationship).

  We require that the key is defined in the topmost class.

Overview

Sources for Information

- Interviews
- Analysis of existing forms

  Filled out forms are more useful than empty ones.
- Analysis of reports which should be generated out of the DB.
- Analysis of existing files
- Analysis of existing manuals, descriptions, documentations (e.g. information for new employees, business reports).
- Partial reuse of published solutions (“reference models”).
- Utilization of experiences of other companies (consultants).
- Standard-Software like SAP/R3 also comes with a DB schema.

Validation

- Use example scenarios. Go over them with the future users.
- If the names of entities and relationships are chosen rigorously, natural language sentences may be produced automatically, which explain e.g. the cardinalities.
  - E.g. “Each and every MUSIC PIECE may be COMPOSED BY one and only one COMPOSER”.
  - Such sentences may be checked by the future users.
- Cross-check database design with the program specification.
  - Does the database provide all necessary information?
  - Is every entity type created/retrieved/updated/deleted by some program (CRUD cycle)?

View Integration

- Often, it is not possible to create the complete ER-Schema in one step, because this would be too large.
- Then you create first small ER-schemas for each application or user. So you start not with the conceptual schema level, but instead with the external views.
- After that, you have to integrate these schemas to get the complete enterprise data model.

  This is not simple: There may be naming conflicts (synonyms, homonyms), type conflicts (the same thing as an entity and as an attribute), domain conflicts (different data types or units of measure), and conflicts among constraints.

Information Needed for Physical Design

Estimation of the size:

- How many entities are there of each type?
- What is the expected growth rate?
- After what time entities can be deleted from the database?
- How long are the string valued attributes typically?
- How large is the percentage of null values?

Estimation of needed performance:

- How often each application will be executed?
- Which applications are especially critical/need fast responses?

  And which can be executed overnight?

Features of Design Tools

- Repository: Database with all design information.
- Graphical Editor for ER-diagrams.
- Version control, views, structuring of large schemas.
- Style and consistency checkers.
- Automatic/Semiautomatic generation of relational tables.
- Automatic/Semiautomatic generation of form-based application programs for entering/viewing data.
- Automatic generation of triggers for integrity constraints.
- Reverse engineering tools, data dictionary browsers.
- Integration with business process models etc.