CPS352 Lecture -The Entity-Relationship Data Modeling; Database Design

last revised January 24, 2017

Objectives:

1. To discuss using an ER model to think about a database at the conceptual design level.
2. To show how to convert an ER design to a relational scheme

Materials:

1. Projectable of various ER diagrams used below
2. Projectable of Book Figure 7.26 p. 309

I. Introduction

A. At the start of the course, we introduced the entity-relationship model as one way of describing a database. We now return to it in more detail. We saw earlier that the entity-relationship model is not, per se, a basis for commercial products; but it is a very useful tool for DESIGNING databases. In particular, we will focus on learning how to picture the conceptual level design of a database using Entity-Relationship (E-R) diagrams.

B. We will also look at how to convert a conceptual design - developed as an ER diagram - into a logical design represented as a relational scheme.

II. Review/Expansion of Definitions - Ask class for definitions

A. Entities and related concepts

1. Entity - an entity is an object that we wish to represent information about.

2. Entity Set - an entity set is the set of all objects of a given kind

   NOTE: Entity sets in a given database do not have to be disjoint. For example, in the library database the same person may be a member both of the entity set employee and of the entity set borrowers

3. Attributes - Individual facts that we store concerning an entity.

   a. Often, the attributes are simple, atomic, single values; but sometimes they may not be. An attribute may be COMPOSITE - i.e. it may have internal structure

      Example: an "address" attribute might be composed of a number, a street, a city, state, and ZIP code

      Note: we have seen that the relational model requires attributes to be atomic. But in doing conceptual design, it may be helpful to think in terms of some composite attributes, even though they will later need to be converted to an atomic form.

   b. For a given entity, a given attribute normally has a single value, but sometimes an attribute needs to be MULTIVALUED

      Example: Some books have multiple authors; we might handle this by making the author attribute of book entities multivalued

      Again: the relational model requires attributes to be single-valued. But in doing conceptual design, it may be helpful to think in terms of multivalued attributes,even though they will later need to be converted to a single-valued form.

   c. Sometimes, a given attribute can be calculated from other information in the database - in which case, instead of storing it we may compute it upon demand. Such an attribute is called a DERIVED attribute.

      Example: As we shall see, the number of books a given borrower currently has checked out can be computed by counting; so we might include books-out as a derived attribute of the borrower entity.

d. Sometimes, we will not know the value of a particular attribute for a particular entity, or it somehow does not apply in a particular case - in which case the value of that attribute is said to be NULL.

4. Domain - the set of possible values for each attribute of an entity is called the domain of that attribute

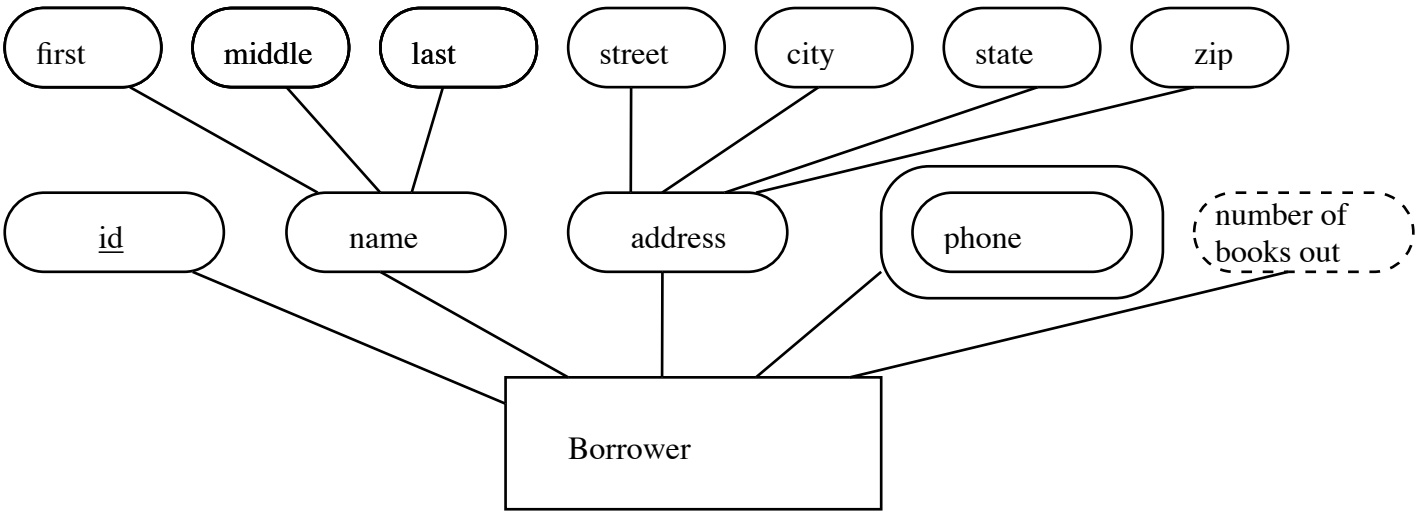
5. Keys: Since the members of a set must be distinct, for any entity set, there must be a set of attributes that serve to uniquely distinguish one entity from all others in the set.

   a. Such an attribute or set of attributes is called a SUPERKEY. Typically, an entity set has many superkeys.

   b. A superkey that has no proper subset that is also a superkey is called a CANDIDATE KEY.

   c. The candidate key chosen by the designer to uniquely identify entities in an entity set is called the PRIMARY KEY. (This is the origin of the term "candidate key" - it is a key that is a candidate for primary key,

6. In an E-R diagram, an entity set is represented by a rectangular box containing the name of the entity set, with its attributes represented by ellipses containing the name of the attribute Each attribute is connected by a line to the entity set.



PROJECT

a. The primary key attribute(s) is/are often underlined.

b. Composite attributes (e.g. name, address) are shown with a hierarchical structure.

c. Multivalued attributes (e.g. phone number) are enclosed in a double ellipse.

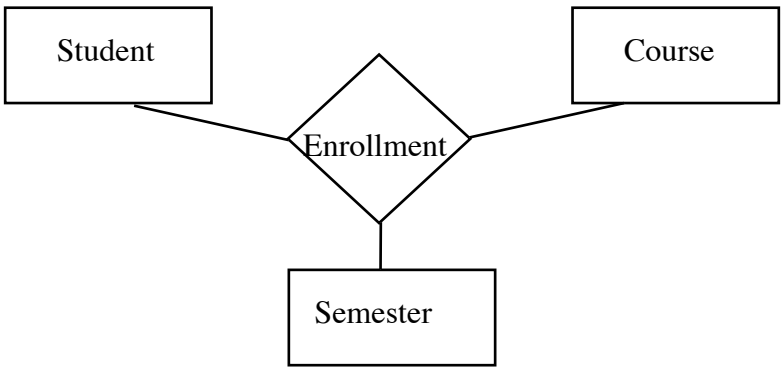
d. Derived attributes (e.g. number of books out) are enclosed in a dashed ellipse.

B. Relationships and related concepts

1. Relationship - a relationship is some connection between two or more entities:

a. Relationships can be binary, ternary, quaternary etc - i.e. they can involve two, three, four or more entities. However, binary relationships (such as "checked out") are by far the most common.

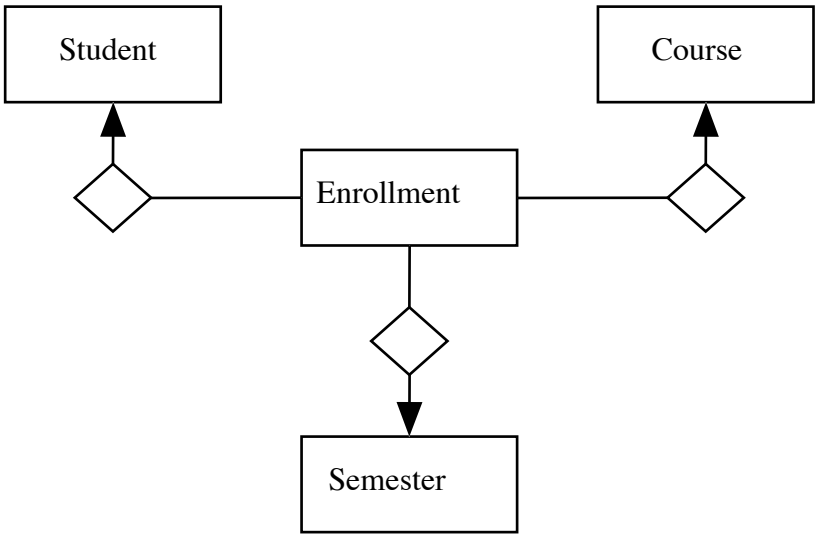
(1) Example of a ternary relationship: student enrollment in a course could be thought of as a ternary relationship involving the student, the course, and the semester.



PROJECT

A relationship set involving n entities (n > 2) can always be converted to a single entity set plus n binary relationships. The original relationship is converted to an entity, connected to each of the other entities through the appropriate set.

Example: Alternate approach to the above.



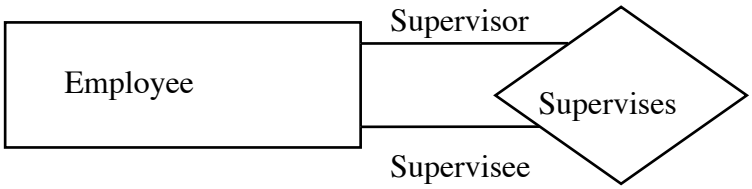
PROJECT

(Note that each of the new binary relationships is "one" on the side connecting to the original entity. Each of the new enrollment entities is associated with exactly one Student, Course, and Semester)

(2) Because a conversion like this is always possible, it is actually rare to see ternary and higher degree relationships in ER diagrams.

b. Normally, the entities in a relationship come from distinct entity sets. Sometimes, though, we can have a relationship in which both entities are from the same entity set. Then, we need to distinguish the role each entity plays in the relationship.

Example: consider the relationship "supervised by" between employees of the library. Both entities in the relationship are from the same entity set (employees), but there are two distinct roles: supervisor and supervisee. In cases such as this, it is common to explicitly include the role names in an ER diagram.



PROJECT

2. Relationship set - a relationship set is the set of all relationships of a given type - just as an entity set is the set of all entities of a given type.

   Example: When a borrower checks out a book, that establishes a relationship between the borrower and the book. The set of all such relationships, together, constitutes the "books checked out" relationship set.

3. Formally, a relationship set is a subset of the cartesian product of the entity sets. The cartesian product is formed by taking every possible combination of elements from the sets.

   Example: Suppose our sets of borrowers and checkouts were as follows:

Borrowers	Books
Anthony Aardvark	Herbs and Shrubs
Ralph Raccoon	Popular Trees
Zelda Zebra Zoo	Guidebook

PROJECT

The Cartesian product of these sets is:

Anthony Aardvark,	Herbs and Shrubs
Anthony Aardvark,	Popular Trees
Anthony Aardvark,	Zoo Guidebook
Ralph Raccoon,	Herbs and Shrubs
Ralph Raccoon,	Popular Trees
Ralph Raccoon,	Zoo Guidebook
Zelda Zebra,	Herbs and Shrubs
Zelda Zebra,	Popular Trees
Zelda Zebra,	Zoo Guidebook

PROJECT

This set represents the set of ALL POSSIBLE RELATIONSHIPS that could ever occur; but obviously the "books checked out" at any one time forms a subset of this set. (Indeed, it could be empty, and - in this particular case - could never have more than three elements since only one person can check out a given book at any one time.)

a. Thus, the join operation of relational algebra - which forms the cartesian product - can be thought of as creating an entity set which includes every possible relationship.

b. Likewise, the natural join operation can be thought of as creating an actual instance of the relationship set corresponding to the current instances of the entity set.

4. There exists a natural physical representation for entity sets: a file of records, wherein each record is an entity and each field an attribute. However, this is not true for relationships; one of the major differences between different types of DBMS's is how they represent relationships.

   a. Hierarchical DBMS's often use physical proximity or pointers to model relationships.

b. Network DBMS's often use circularly linked lists for relationships

c. OO Databases may use references or pointers to model relationships.

d. Relational DBMS's do not distinguish between entities and relationships; a relationship is just another entity. Thus, the same physical representation is used for both. Note that the handling of relationships represents a key difference between relational databases and the other kinds of systems.

5. Like entities, relationships can have attributes (called descriptive attributes) associated with them.

a. Note that we want to associate an attribute with a relationship just when it is a property of the relationship, not of the participating entities - e.g. in the case of date due for a book:

(1) It makes no sense to say that "date due" is a property of a borrower. A borrower may have several different books out, with different dates due.

(2) A date due is not really a property of a book, in the sense that a given book may be checked out by different borrowers at different times, with different dates due.

b. This leads to a design question. Sometimes, we must choose between modeling a given type of information by:

• A relationship with attributes

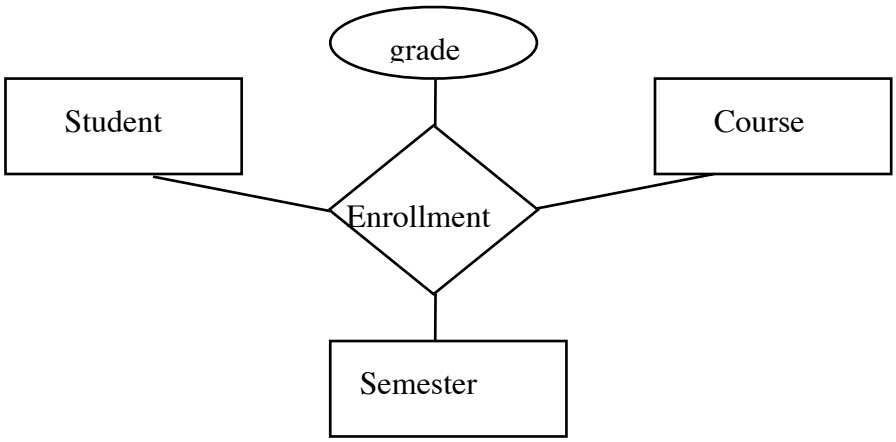
• A relationship without attributes, plus additional attributes in one of the entities related.

• A new entity

Example: the relationship "checked-out" between borrowers and books has the attribute date_due. This could be modeled alternately by a relationship without attributes plus a current date_due attribute for book (assuming we had no need to store past dates due, since a book can only be checked out to one person at a time) or by a new checked-out entity with date_due attribute related to a borrower and a book. Depending on how we plan to use the database, one or the other of these may be preferable.

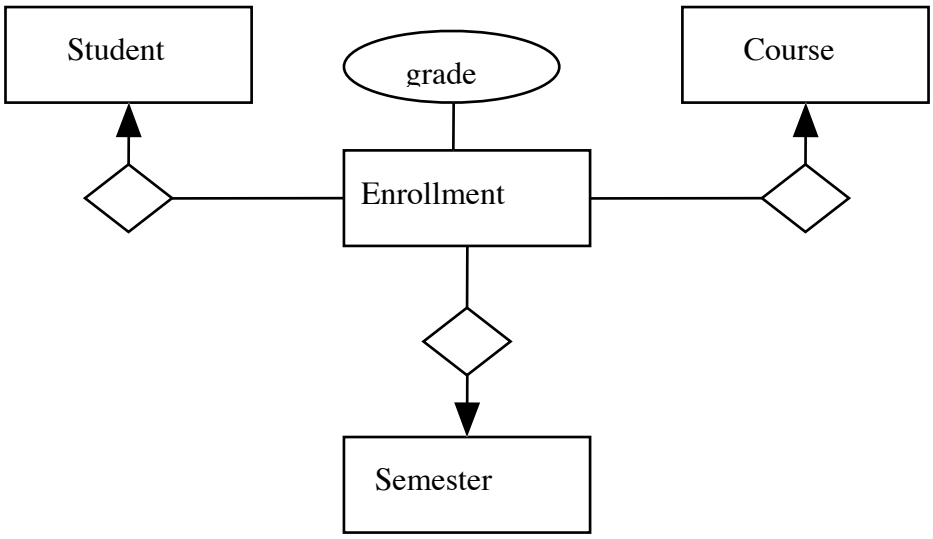
c. When we convert a ternary or higher degree relationship to a binary one by replacing the relationship with an entity, any attributes of the relationship become attributes of the entity that was created to replace it.

Example: we used an example earlier of a ternary relationship between Student, Course, and Semester. Suppose this relationship has a grade. This is properly an attribute of the relationship (if the student takes the course twice, each Semester has its own grade.) Thus we have:



PROJECT

which we convert to



PROJECT

(In fact, all of the models except the relational model have trouble representing relationships with attributes anyway - so it is quite common to convert even binary relationships to entities if they have attributes.)

C. Mapping Constraints - We have seen that a relationship set is a subset of the cartesian product of the entity sets it relates. Often, the logic of the data being modeled will impose some restrictions as to what kind subsets are possible.

Example: We would not expect to find the following in the "books checked out" relationship:

Anthony Aardvark,	Herbs and Shrubs
Zelda Zebra,	Herbs and Shrubs

Why?

ASK

1. Mapping Cardinalities - the most common constraints are mapping cardinalities.

a. In general, a binary relationship can be

   (1) One to one - the most tightly constrained

   (2) One to many or many to one

   (3) Many to many - the least constrained

   Example: The relationship set "checked out" is one to many from borrowers to books - a borrower can have many books checked out, but a book can only be checked out to one borrower at a time.

b. If a given relationship set is one to one, any member of either entity set involved in the relationship in question can participate in at most one relationship of the kind in question.

   Example: heterosexual marriage is a one to one relationship between the entity sets men and women

   Note that a one to one mapping constraint for a given relationship set does not imply that a given entity MUST participate in a relationship of that kind - only that it CAN participate in at most one such relationship. (The relationship marriage allows for bachelors and bachelorettes, widows and widowers etc.)

c. If a relationship set is one to many, then any entity in the first entity set can participate in any number of relationships, but an entity in the second set can participate in at most one.

   Example: as we have already noted, the relationship "books checked out" is one to many from borrowers to books.

d. If a relationship set is many to one, then any entity in the first entity set can participate in at most one relationship, while entities in the second entity set can participate in any number.

   Example: the relationship "supervised by" may many to one from employees (in the supervised by role) to employees (in the supervisor role.) Each employee has at most one supervisor; but a supervisor can supervise many employees.

Of course a one to many relationship from A to B is a many to one relationship from B to A - i.e. there is no fundamental difference between the two concepts - its just a matter of how we refer to it.

e. If a relationship set is many to many, then any entity of either set can participate in any number of relationships.

Example: Suppose we defined the relationship "has taken out" between borrowers and books, which records every book a given borrower has ever taken out (whether or not he has it out now.) (This is actually a very bad idea on privacy grounds, of course - but allows us to illustrate a point about database design!) This is many to many, since:

- Any given borrower can take out any number of books
- Over time, any given book can be checked out by any number of borrowers

f. Note that one-to-one and one-to-many / many-to-one relationships are most common. In fact, as we have seen, the hierarchical, network, and object-oriented models of all have some difficulty with many-to-many relationships, and the relational model can often use a much more efficient representation for one-to-one and one-to-many relationships than what must be used for many-to-many relationships.

g. Primary keys for relationship sets - just as any strong entity set has a primary key, so does any relationship set. The primary key of a relationship set depends on its cardinality.

(1) If it is many-to-many, then its primary key is simply the union of the key attributes of the entities it relates.

Example: Suppose we want to record all borrowers who have ever checked out a given book. Since the primary key for borrowers is

borrower-id, and that for books is call number + copy number, the primary key for has ever checked out is the three attributes borrower id, call number, and copy number together.

Note: Since this is a relationship set, we will record the fact that a given borrower has checked out a given book just once in it, even if the same borrower takes out the same book again.

(2) If it is one to many or many to one, then its primary key is the primary key of the "one" entity.

Example: A book can only be checked out to one borrower at a time, but a given borrower can check out many books - so checked out is many to one from books to borrowers. Suppose the primary key for books is call number + copy number. Then the primary key for checked out, like that for books, is call number, and copy number together.

(3) If it is one to one, then the primary key is the primary key of *either* of the entities.

h. Mapping cardinalities are often shown in E-R diagrams by the use of an arrowhead pointing to the "one" entity(s) - i.e.

- In a one to one relationship, arrowheads point to both entities
- In a one to many relationship, an arrowhead points to the first
- In a many to one relationship, an arrowhead points to the second
- In a many to many relationship, the diagram contains no arrowheads.

(Think of the arrowhead as indicating that we can define a function that, given a book, returns its borrower, if any. The lack of an arrowhead going the other way says we cannot define a function that, given a borrower, returns the book (s)he holds, since he can hold many books and by definition a function is single-valued.)

Note: notation is not consistent in this regard - some authors have the arrow going the other way! Moreover, as the text points out, it is possible to use numbers (like 1..n) instead of arrows, but the convention in ER diagrams is the exact opposite of that in UML class diagrams - very confusing! - so we will avoid this.

2. Participation constraints: in some cases, the underlying reality dictates that every entity in one of the entity sets must participate in an instance of the relationship.

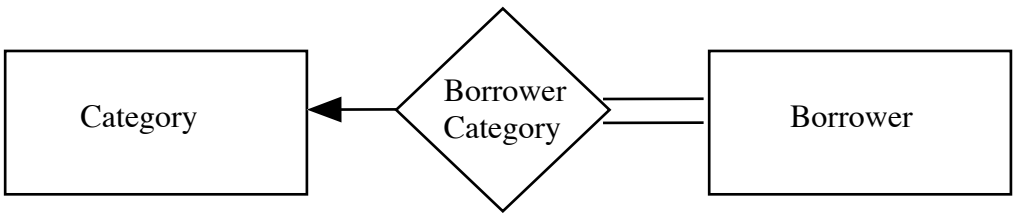
   Example: Suppose we have the notion of a borrower category, with different categories of borrowers allowed different privileges such as length of time to keep a book out. (This is, in fact, the case with the Gordon library - in fact, a faculty borrower used to be able to keep a book out for a year.)

   We might want to require that every borrower be related to some category.

   a)This is called a total participation constraint (as opposed to partial participation.

   b)It is represented in an ER diagram by using a double line for the connection between the relationship and the entity that must participate.

   Example:



PROJECT

(Note: this diagram says each Borrower must not only participate in an instance of the relationship, but must participate exactly once - it would not make sense to have a borrower be simultaneously in two or more categories!)

3. Existence dependencies - another form of constrained relationship, wherein the existence of an entity in one entity set is dependent on the existence of an entity in the other entity set.

   a. We said earlier that - in almost every case - the complete set of attributes for an entity is a superkey for the entity set.

      (1) In many cases, this is uninteresting, though there are some entity sets for which the full set of attributes is the only possible superkey.

      (2) There is a certain kind of entity set - called a *weak* entity set - in which even the full set of attributes may not be sufficient to be a superkey. (There is no superkey for a weak entity set.) A weak entity is always subordinated to some other strong entity, on whose very existence it depends. (More about this later).

      Example: In a library database, a fine can be represented as a weak entity depending on the borrower entity who owes it, if the fine entity only stores the amount.

      We call the entity whose existence depends on another SUBORDINATE, and the other entity DOMINANT.

   b. The key property of an existence dependency relationship is that if a dominant entity in the relationship is deleted from the database for some reason, then all subordinate entities depending on it must also be deleted

Example: if the borrower owing some fine is removed from the database, we must also remove the record of the fine owed. [An off-campus borrower who moves out of state can get away with leaving outstanding fines behind him that can never be collected.]

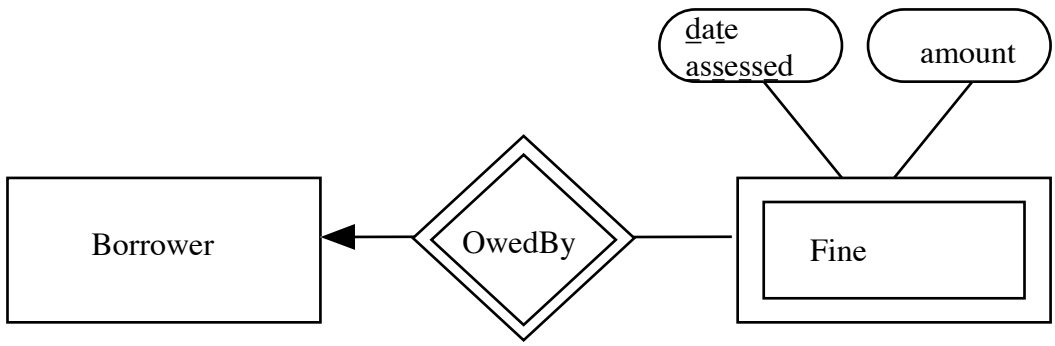
c. Entities within a weak entity set are uniquely identified by the combination of some or all of their own attributes plus the entity on which they depend.

Example: Suppose that the library assesses all fines once a day, lumping together into one amount all the amounts owed for all overdue books returned that day by a given borrower. Then a given fine - represented as amount due and date assessed - can be uniquely specified by specifying the date it was assessed plus the borrower who owes it.

d. The attributes of a weak entity that identify it uniquely among all the weak entities related to a given strong entity are called its DISCRIMINATOR or PARTIAL KEY. The primary key of a weak entity consists of its partial key plus the primary key of the entity on which it depends.

e. In an ER diagram, a weak entity is represented by a double box, and the corresponding existence-dependency relationship by a double diamond. The discriminator attributes of the weak entity are underlined using a dashed line.

Example (attributes of borrower omitted for simplicity)



PROJECT

III. **Generalization and Specialization**

A. Sometimes, a given entity set may contain several distinguishable groups, with some attributes in common and some distinct to each group.

EXAMPLE: The entity set borrowers may be composed of student borrowers, faculty/staff borrowers, and community borrowers. All borrowers have a name and address. Student borrowers have a student id and a class year (freshman, sophomore etc.). Faculty/staff borrowers have a faculty/staff id and a campus department they are employed by. Community borrowers may have a special id generated just for library use, plus some relationship to the college that explains why they are granted borrowing privileges here (e.g. NECCUM affiliation.)

B. This kind of situation - and some of the nuances connected with it - is discussed in the book. If it sounds a lot like inheritance in OO, that's because that's really what it is!

IV. **Representing Entities and Relationships by Relational Tables**

A. As we have already noted, any database scheme consisting of entities and relationships can be represented by a series of TABLES - one for each entity set, plus one for each relationship set. These often correspond to the relations of a relational database, though in some cases some adjustments need to be made.

B. A strong entity set can be represented by a table with one row for each entity, and one column for each attribute. When we write such a table out, we often label the columns with the names of the attributes, or an abbreviation for them.

EXAMPLE: The entity set borrowers - with attributes borrower_id, last_name, first_name, - might be represented by:

ID	last_name	first_name
12345	Aardvark	Anthony
20174	Cat	Charlene
...		

PROJECT

C. A weak entity set can be represented similarly - except that we add a column or columns containing the primary key(s) of the strong entity(s) on which the weak entity depends:

EXAMPLE: if the entity set fines - with attributes amount owed and date assessed - is a weak entity set subordinate to borrower, and the primary key of borrowers is borrower-id, then fines can be represented as follows:

borrower_ID	amount	date_assessed
12345	$0.25	12-1-16
12345	$0.50	12-2-16
20174	$0.10	11-15-16
...		
PROJECT		

D. A relationship set can be represented by a table with one row for each relationship, and with one column for each of its own attributes, plus one column for each primary key attribute of each entity participating in the relationship.

EXAMPLE: If the checked-out relationship has attribute date-due, and relates borrowers and books, and the primary key of borrowers is borrower-id, and of books is call-number plus copy-number, then checked-out can be represented as:

borrower_ID	call_number	date_due
89754	RZ12.905	11-10-16
89754	LM925.04	11-10-16
...		
PROJECT		

E. If the relationship is one-to-one or one-to-many, it is also possible to “fold” it into the “many” entity (by including the foreign key of the “one” entity and any attributes), with the understanding that these will be null for an entity that is not in any relationship.

EXAMPLE: since checked_out is one to many from borrowers to books, it could also be represented by folding it into the book entity as follows:

call_number	title	author	borrower_id	date_due
QA76.093	Wenham Zoo Guide	elephant	null	null
RZ12.905	Fire Hydrants I Have Known	dog	89754	11-10-16

PROJECT

F. Where two entity sets are related by generalization or specialization, we have two major options:

1. We can use one table for each specialized entity set, containing all the attributes of that set. No table need be used to represent that entity set which represents their generalization: the information can be constructed when needed by combining the specialized tables.

   EXAMPLE: Suppose that all borrowers of the library are either students, faculty_staff, or community borrowers. We might use three separate tables, with no one table containing all the borrowers - e.g.

   Student_Borrowers:

student_ID	last_name	first_name	class_year

   Faculty_Staff_Borrowers:

employee_ID	last_name	first_name	department

   Community_Borrowers:

borrower_ID	last_name	first_name	relationship

   In this case, we have to take a union of the three tables if we want to get a complete list of borrowers.

2. We can use one table to represent all of the attributes that the special groups have in common, plus one for each special group holding its unique attributes, plus the primary key from the general table.

   EXAMPLE: Suppose we ensure that student ids, employee ids, and community borrower ids have different formats, so that no member of one domain can ever be a member of another. (E.g. student ids are all numbers, employee ids begin with the letters FAC or STF, and community borrower ids begin with letters COM). Then we can represent our entity sets as follows:

   Borrowers
   ID name address

   Student_Borrowers
   ID class_year
   Faculty_Staff_Borrowers:
   ID department
   Community_Borrowers:
   ID relationship

3. Note that the choice here can depend on whether a "general" entity must be a member of some specialized group or not, and whether the "general" entity can be a member of more than one specialized group. If the "general" entity must be a member of exactly one specialized group, then the former approach can be used; otherwise, the latter approach must be used.

V. **E-R Modeling and UML**

A. The E-R diagrams we have been using are actually a precursor to the UML class diagrams we learned about in CPS122. Of course, OO class diagrams are quite different from E-R diagrams. It might be convenient if everybody used just one type of diagram, but that's not the case.

1. UML diagrams are typically used in the design of OO software

2. E-R diagrams are typically used in the design of database schema.

3. A given system that has an OO "front-end" and a relational database "back-end" may, in fact, be diagrammed by using both kinds of diagrams in the different contexts! (This has happened with some senior projects).

B. The book contains a diagram that illustrates some of the key differences between the two notations. Note the close correspondence between the E-R notion of "entity" and the OO notion of "class"

   PROJECT: Book Figure 7.26 p. 309

C. Some key differences:

1. In E-R diagrams, attributes are typically shown outside the symbol for an entity; in a class diagram they are shown inside the class box. (Though an earlier form of class diagram from a precursor to UML used the E-R style notation.)

2. In E-R diagrams, relationships are always depicted by diamonds connected to the participating entities by lines; in UML diagrams, they are represented by simple lines connecting the participating classes - unless the relationship has attributes, in which case a relationship class is needed.

3. In E-R diagrams, an alternate notation for cardinality is to explicitly specify it using notations like 0..1, 1..1, 0..n, or 1..n, or more specific values. **But note well** the E-R convention is the opposite of the UML convention as to where the numbers go!

4. Different notations are used for generalization/specialization, which is essentially UML inheritance.