CS352 Lecture - The Entity-Relationship Model

last revised August 3, 2004

Objectives:

1. To introduce the concepts of “entity”, “relationship”, “key”
2. To show how to convert an ER design to a set of tables.

Materials:

1. Transparency: E-R and UML notation - textbook page 69

I. Introduction

A. We have seen that a database management system typically describes a database at three levels of description.

1. The physical level - how the data is stored in files

2. The conceptual level - the "big picture"

3. The view level - individual views of the database for each application

B. In order to be able to describe a database, we need some system of notation and representation - a data model. This is true at all levels; but, we are particularly concerned with description at the conceptual and view levels. We must describe two things:

1. Data objects

2. Relationships between data objects

C. We will be learning about five different data models that can be used in designing and implementing databases (listed here in the order we will cover them, which is NOT the historical order of development):

1. The entity-relationship model

2. The network model

3. The hierarchical model

4. The relational model

5. Various OO models and XML (there is no single established one)

The last four have been and are used as bases for commercial DBMS's.

D. The entity-relationship model is not, per se, a basis for commercial products; but it is a very useful tool for DESIGNING databases. Also, once the E-R model is understood it gives us a language we can use in talking about the other models. Thus, we start with it. In particular, we will focus on learning how to picture the conceptual level design of a database using Entity-Relationship (E-R) diagrams.

E. The textbook used a banking enterprise as a source of examples. We will use a college library for our examples today.

II. Definitions

A. Entities and related concepts

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

   Example: books, periodicals, borrowers, late fines owed, employees

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

   Example: a single book is an entity, that is a member of the entity set comprising the library's holdings

   a single borrower is an entity, that is a member of the entity set comprising all the people allowed to borrow books from the library

   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.

   Example: Some attributes we might be interested in for a book are:

   Author (assume just one for simplicity)
   Title
   Copyright date
   Call number
   Copy number (allows multiple copies of a given book)
   Accession number (This is like a serial number assigned to a book at the time of purchase.)

a) Note, that, in general, we do not choose to store every possible fact about an entity, but only those of interest to us for some useful reason. For example, a library database would probably not contain an attribute for each book giving the color of its binding - though it could if this were desired.

   Example: For a fine, we might store

   Amount owed
   Date assessed

b) 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

c) 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

   Note: Because the relational data model does not allow composite or multivalued attributes, we may need to develop an ER design for a relational database with the knowledge that we will have to handle them differently in the relational implementation. Using composite or multivalued attributes in the conceptual design may still make sense.

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. (This has a bit of a different meaning here than when used in the phrase “null pointer”)

   Example: One attribute we may wish to store for a book is its copyright date. But some books (e.g. government publications) are not copyrighted, and so this attribute would be NULL for such books.

   (1) Nulls can arise because a particular attribute is not applicable to a given entity.

(2) Nulls can also arise because a value may be unknown - e.g. the phone number of a borrower may be unlisted.

e) 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.

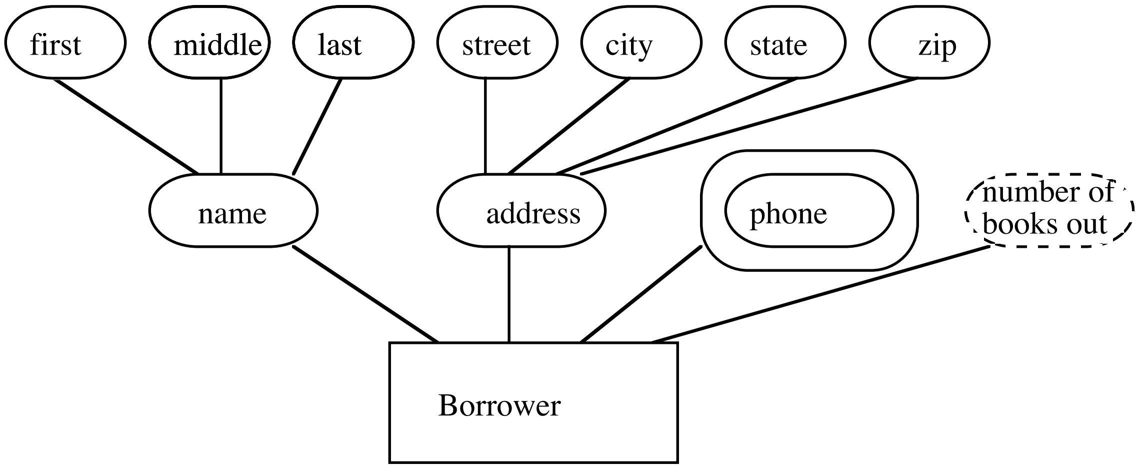
Example: As we shall see, the number of books a given borrower currently has checked out can be computed by counting; so we might include books-out as a derived attribute of the borrower entity.

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

Example: The domain of the book attribute Publisher is the set of all companies that publish (or have published) books.

The domain of the fine attribute date assessed is the set of all valid dates. (We would not expect to find Dec 32, 2003 as a value for this attribute.)

5. 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.



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

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

c) 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:

   Example: We could define the "checked out" relationship as a relationship between a borrower and a book

   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.

      Example of a ternary relationship: if a borrower checks out a book and returns it late, and thus is assessed a fine, there exists a three-way relationship between the borrower, the book, and the fine.

   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.

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 books were as follows:

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

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

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.)

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.

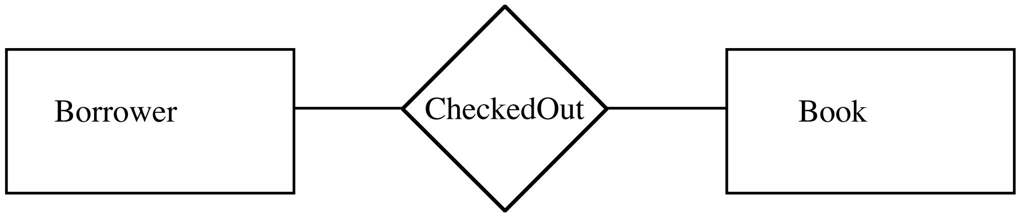
   b) Network DBMS's often use circularly linked lists for relationships

   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.

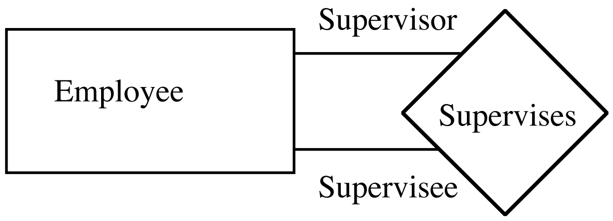
5. Relationship sets are represented in E-R diagrams by a diamond, with lines connecting it to the entity sets involved.

   Example: checked out (attributes omitted from entity sets for brevity)



6. Where it is important to distinguish the roles played by entities in the relationship, the connector line can be labeled with the role name. This is especially important in the case of *recursive relationships*, in which some entity set is related to itself.

   Example: Supervision:



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

   Example: For the checked out relationship between borrowers and books, we might have the attribute date due.

   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.

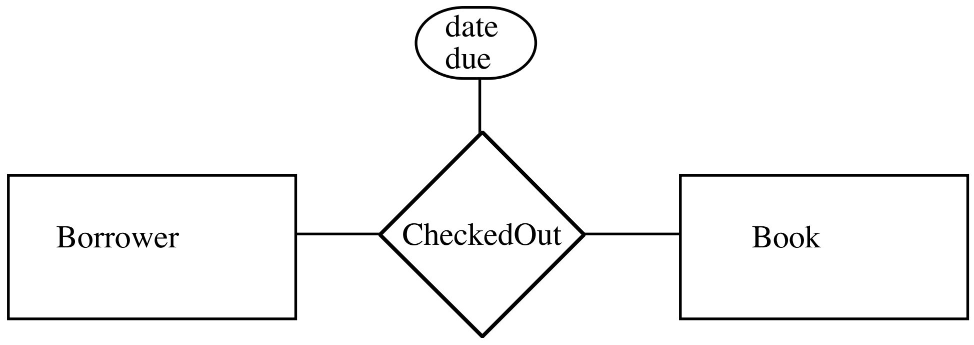
      In the case of date due for a book:

      (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 can be pictured in an E-R diagram by attaching an attribute to the relationship diamond.

Example: The above - only the relationship attribute shown for brevity:



c) 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 <u>current</u> 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. 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

   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: marriage is a one to one relationship between the entity sets men and women (at least in most Western countries)

      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: the relationship "books checked out" is one to many from borrowers to books. One borrower can check out many books; but any given book can be held by at most one borrower at any one time.

   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.

Note that 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.

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

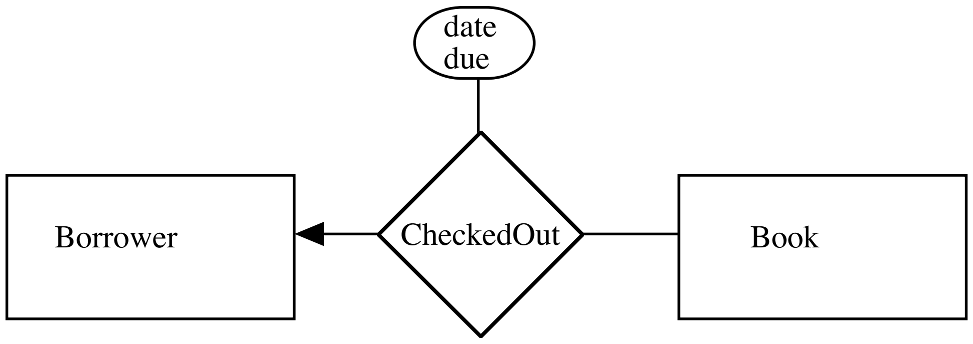
• In a one to one relationship, arrowheads point to both entities

• 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.

Example: Checked out is one to many from borrowers to books. This can be shown as follows (omitting attributes):



(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 class diagrams - very confusing!

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

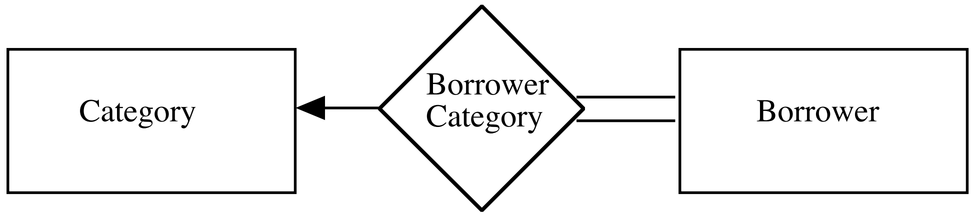
   Example: Suppose we have the notion of a borrower category, with different categories of borrowers allowed different privileges such as length of time to keep a book out. (This is, in fact, the case with the Gordon library - a faculty borrower can keep a book out for a year.)

   We might want to require that <u>every</u> borrower be related to some category.

   a) This is called a <u>total participation</u> 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:



(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.

   Example: We suggested we might include in a library database an entity set of late fines outstanding. Each such fine, of course, is connected to the borrower who owes it by a relationship we might called "owed by". This relationship is, in fact, an existence dependency.

   (We will say more about this in conjunction with the notion of weak entities shortly.)

   a) 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.]

D. Keys - We have said that an entity set is a set, in the mathematical sense - that is, each of its members must be distinct. This implies that the members of an entity set must be DISTINGUISHABLE - there must be some difference among them whereby we can tell them apart.

1. We expect that the value(s) of a single attribute or a group of attributes will suffice to distinguish one member of an entity set from all others

   Example: One attribute typically stored for an employee is a Social Security Number. This attribute distinguishes one employee from another - there can never be two different employee entities with the same SSN attribute. (At least there aren't supposed to be, though the system has been known to mess up!)

   Example: For a book, we may store a call number and a copy number. Neither, by itself, distinguishes a book from all others - we may have several copies of a given book, and we certainly have lots of books that are "Copy 1" of their title. However, the combination of call number + copy number, together, is unique.

   Note that this actually represents a significant difference between the "database approach" and the "OO approach" to representing informaton. In the "database approach", <u>identity</u> is established by <u>value</u> - two distinct entities must have values that are distinct in some way. In OO systems, identity is often established separately from value - i.e. two objects might have distinct identities even though they happen to contain the same values. [More on this when we get to the discussion of OO databases].

2. Superkey - We call any set of attribute values which suffices to distinguish one member of an entity set from all others a superkey for that entity set.

   a) In general, there can be many superkeys for a given entity set.

      Example: For a book, the call number + copy number, together, would be a superkey.

      For a book, the accession number, by itself, would also be a superkey.

      Question: Give another potential superkey for a book (not a superset of one of the above.)

      ASK

   b) If a given set of attributes is a superkey, then any superset of those attributes is also a superkey.

Example: call number + copy number for a book is a superkey. Therefore, all of the following are also superkeys for a book:

call number + copy number + author
call number + copy number + title
call number + copy number + edition
...

c) Note 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 we will encounter some entity sets for which the full set of attributes is the only possible superkey.

(2) We will see shortly that for a certain kind of entity set - called a *weak* entity set - even the full set of attributes may not be sufficient to be a superkey. (There is no superkey for a weak entity set.)

3. Candidate key - a candidate key is a superkey which has no proper subsets that are also superkeys - i.e. it is, in some sense minimal.

Example: For books, accession number is a candidate key, as is call number + copy number. But sets of attributes like

accession number + copyright date
call number + copy number + author
accession number + call number

though superkeys, are not candidate keys.

4. Note that the notions of superkey and candidate key are determined by the logic of the database scheme, not by a particular instance.

Example: suppose that, at a certain point in time, a given small library had no two borrowers with the same last name.

Last name would not be a superkey for the entity set borrowers, because there is no reason in the logic of the enterprise why this should always be true.

Again, in this case the superkey last name + first name + middle initial would contain a proper subset (last name alone) that suffices to uniquely identify a borrower. But this would not prevent last name +

first name + middle initial from being a candidate key, since last name is not guaranteed to always be able to function as a superkey.

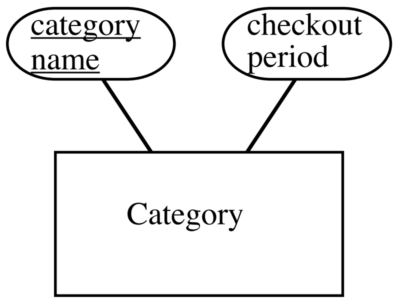
5. Primary key - the primary key of an entity set is a particular candidate key chosen by the database designer as the basis for uniquely identifying entities in the entity set.

   a) Candidate keys are called that because they are candidates for being chosen as the primary key.

   b) Generally, the primary key is chosen to be the candidate key that is, in some sense, simplest. Thus, if we have a borrower ID, we will normally prefer that as the candidate key, rather than using last name + first name + middle initial + address.

   c) It is standard practice, in an ER diagram, to underline the attributes comprising the primary key.

   Example:



6. We mentioned earlier the notion of a weak entity as one which has no superkey.

   Example: The entity set fines owed contains attributes amount and date assessed. There is no good reason why there could not be many entities in the set of current outstanding fines assessed on the same day for the same amount - in fact, we expect there probably would be.

   a) All weak entities have an existence dependency on some other entity.

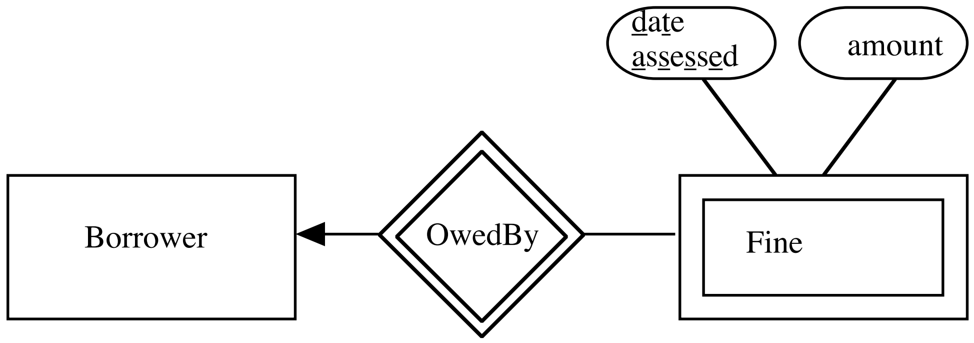
   b) Entities within a weak entity set are uniquely identified by the combination of some or all of their own attributes plus the entity on which they depend.

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. (This makes the relationship between fines and borrowers binary rather than ternary as in the a previous example.) 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.

c) 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.

d) 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)



7. 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.

a) 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.

b) 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.

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

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 Tables

A. 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.

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, address, and telephone - might be represented by:

ID	last_name	first_name	address	telephone
12345	Aardvark	Anthony	Jenks subbasement	x9876
20174	Cat	Charlene	Frost Basement	x9875
...				

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: The entity set fines - with attributes amount owed and date assessed - is a weak entity set. Each entity in it depends on a borrower entity. If the primary key of borrowers is borrower-id, then fines can be represented as follows:

borrower_ID	amount	date_assessed
12345	$0.25	12-1-02
12345	$0.50	12-2-02
20174	$0.10 0	11-15-02
...		

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: Suppose the checked-out relationship has attribute date-due, and relates borrowers and books. Suppose further that 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	copy_number	date_due
12345	QA76.0379	2	11-29-02
12345	QA76.1234	1	12-19-02
...			

E. 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	name	address	class_year

Faculty_Staff_Borrowers:

employee_ID	name	address	department

Community_Borrowers:

borrower_ID	name	address	relationship

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 depends 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.

F. When we come to relational model later in the course, we will see that it is based on precisely this method of representing entities and relationships by means of tables. This is a very general and elegant approach, but does have some performance implications. The other models we will consider (hierarchical, network, and object-oriented) use a form of table for entities, and use pointers to represent relationships. This is a less general and elegant scheme - but does have performance advantages.

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 CS112 and CS211. 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”

TRANSPARENCY: Book page 69

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.