CPS352 Lecture -The Entity-Relationship Model last revised February 2, 2021

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. Projectables of various ER diagram notations and one showing all together.
2. Projectable of Borrower ER diagram showing various kinds of attributes (drawn both Chen notation and notation used in text.)
3. Projectable of ternary relationship
4. Projectable of recursive relationship
5. Projectable of Cartesian join
6. Projectable of total participation constraint
7. Projectable of weak entity
8. Projectable of converting and ER diagram to a relational schema.
9. Saved file - RegistrationExample.xml - for demoing tool

I. Introduction

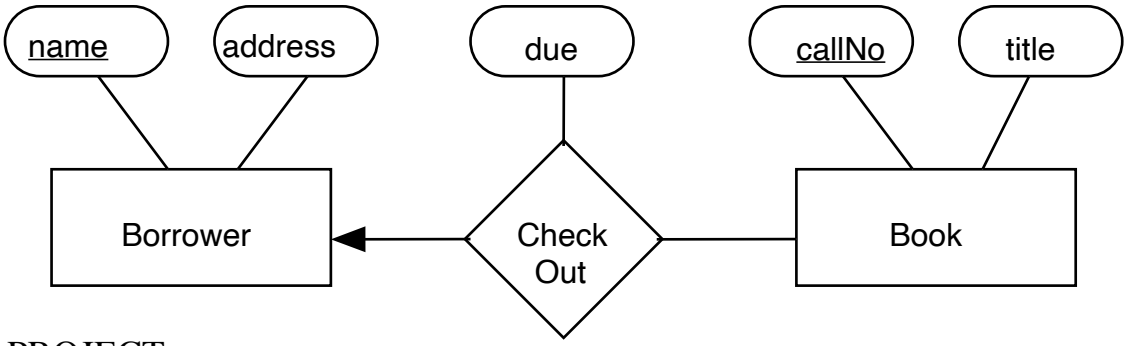
A. In the last series of lectures, we introduced the entity-relationship model as one way of describing a database. We now return to it in more detail. We noted that the entity-relationship model is not, per se, a basis for commercial products; but it is a very useful tool for DESIGNING databases. In 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.

C. An ER diagram consists of two basic things - entities and relationships.

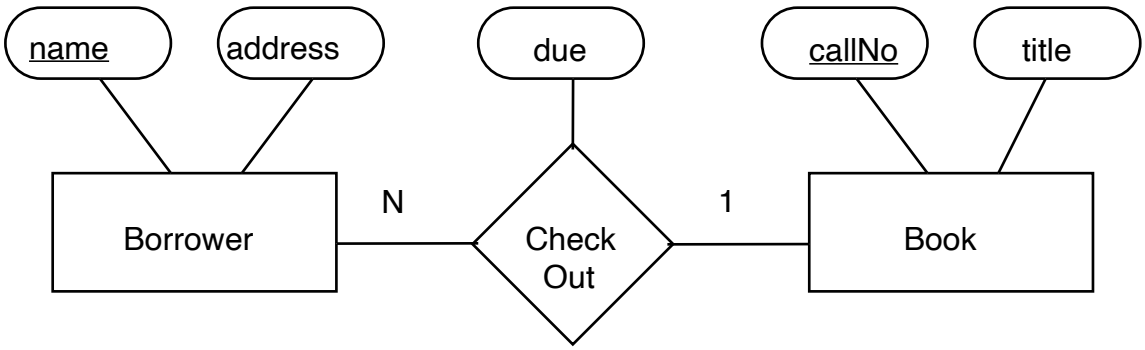
D. However, there are several different notation systems that are used for representing these in ER diagrams. We can illustrate this by showing several different ways of drawing a diagram representing a simplified version of a portion of the library database we used in the previous lecture.

1. The original notation scheme - called "Chen" notation after its inventor - Peter Chen. This is the notation most commonly used in textbooks for a course like this - including earlier editions of the book we are using. The arrow connecting the CheckOut relationship to borrower indicates that a given book may only be checked out to one borrower at a time - i.e. given a book you can ask for the one and only borrower (if any) to whom it is checked out. We call this a one to many relationship.



PROJECT

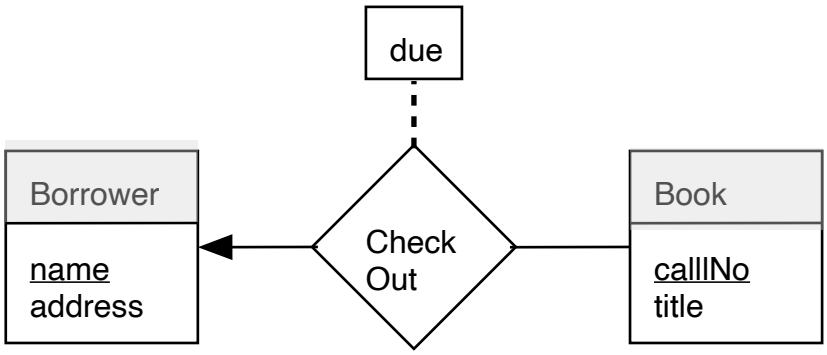
or, using a different representation for multiplicity of relationships, likewise indicating that a borrower can have any number of books out, but a given book can only be checked out to one borrower at a time. (Sometimes * is used in place of N - or M for a many-many relationship).



PROJECT

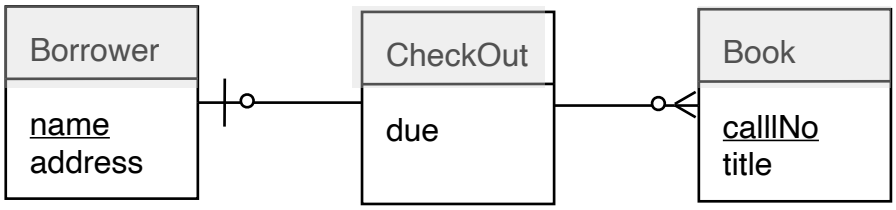
Important note: placement of the cardinalities is opposite to usage in UML! This diagram says that each Book is checked out to (at most) one borrower - not that each borrower has one book! (To avoid this confusion, we will prefer the "arrow" style.)

2. A variant of Chen notation used in our textbook. The attributes are listed in the boxes for entities and those for relationships in boxes using "UML--style" rather than in bubbles, which is more compact,



PROJECT

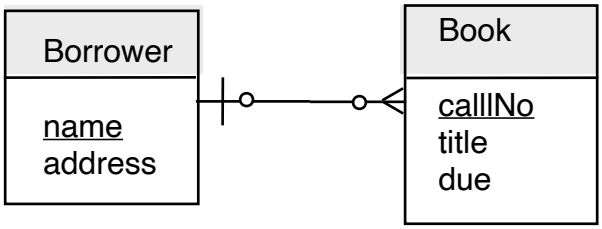
3. A system known as "crows foot" notation. Note the symbols used to denoted "one" and "many". The small circles indicate that the relationship is optional on both sides - a book does not need to be checked out, and a borrower does not need to have any books checked out.



PROJECT

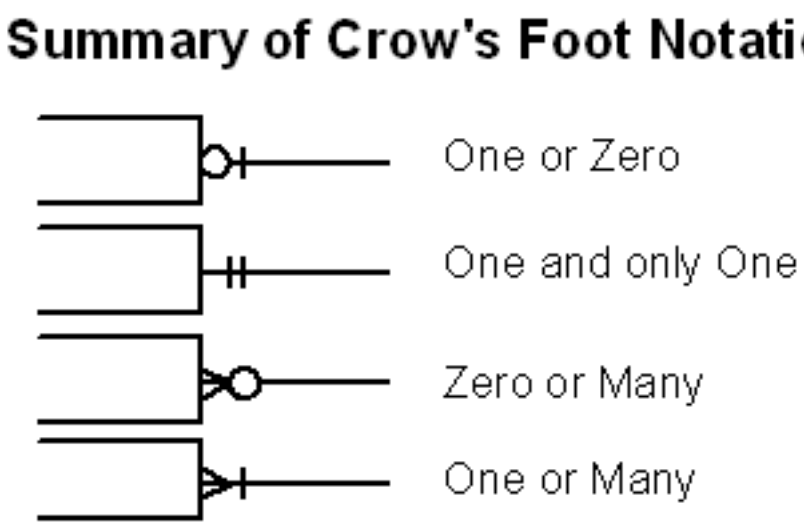
a. Actually, this example misses one important thing about crow's foot notation - relationships are usually **not** represented separately unless they have attributes of their own.

b. In this particular case, we could fold the due date into the Book entity since a book can only be out to one person at a time. This eliminates the need for treating CheckOut as an entity, and so the diagram becomes this:



PROJECT

c. Here is a summary of the notation for cardinality of relationships:



REFER TO IN LAST PROJECTABLE

d. Crow's foot is actually the most-commonly used notation in industry, and seems to be the one most often supported by professional diagramming tools.

4. PROJECT VERSION WITH ALL FIVE TOGETHER FOR COMPARISON

5. For the rest of this lecture, we will generally stick with the form used in the book for consistency with what you are reading, and this will be the style you should use in the design project and homework (hand-drawn if you don't have an appropriate drawing tool to use.). (Actually, in many cases we will omit attributes totally to focus on one aspect of the example!)

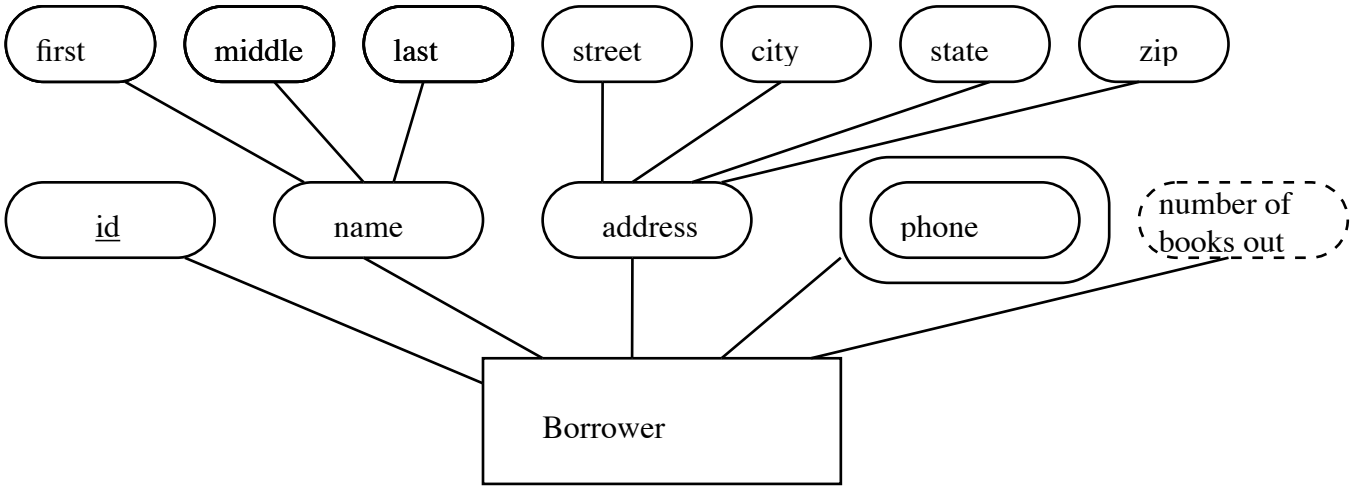
II. Expansion of Definitions

A. Entities and related concepts. We will illustrate with a single entity for now.

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

3. Attributes - Individual facts that we store concerning an entity. To illustrate the various possibilities, we will use the following diagram:



PROJECT DIAGRAM WITH TWO VERSION AND LEAVE UP

Borrower
id name first middle last address street city state zip [phone] number_books_out ()

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: The example shows that the "name" attribute is composed of a first, middle, and last name; the "address" attribute is composed of a street, a city, state, and ZIP code. Note how this is shown in Chen notation and in notation used in book.

   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: A person may have multiple phone numbers. Again, note how this is shown in Chen notation and in notation used in book.

   Note: 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. Note how this is shown in Chen notation and in notation used in book.

   Example: As we shall see, the number of books a given borrower currently has checked out can be computed by counting whenever the value is needed; so we might include books-out as a derived attribute of the borrower entity, though no value for this stored explicitly in the database.

   (In the notation in the book, this would be represented as a separate function)

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. (There is no special diagrammatic notation for this, since it is just a value for particular entity).

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. Recall our discussion of these in connection with the relational model.

   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.) The primary key attribute(s) for an entity are underlined, as is the case for id in Borrower. (Sometimes this is a bit hard to see - when drawing by hand be sure it is clear!)

   d. Some kinds of entities - called weak entities - don't have any primary key. We will look at an example and how this is handled below.

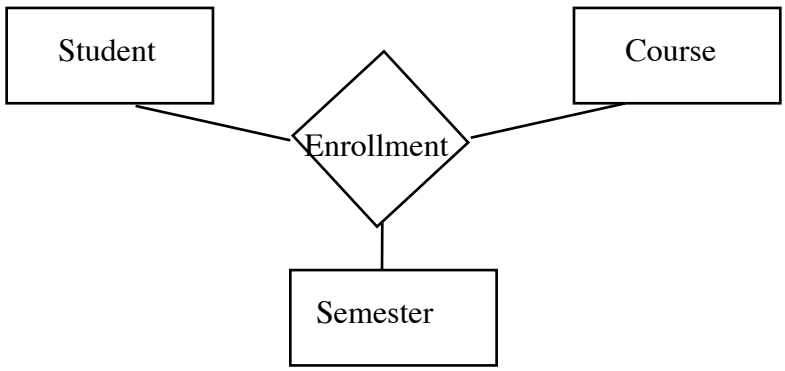
B. Relationships and related concepts

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

   a. A relationship is denoted by a diamond in most notations, as in the examples we looked at earlier.

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

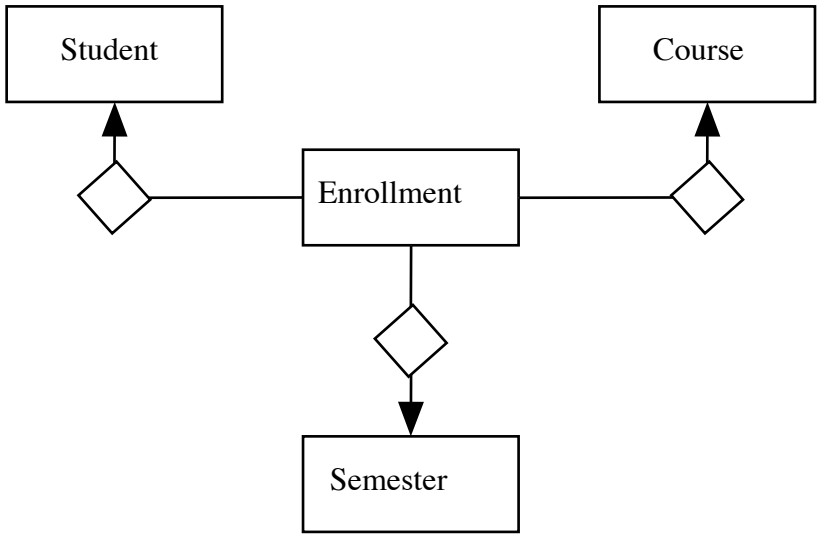
      (1)Example of a ternary relationship: student enrollment in a course could be thought of as a ternary relationship involving the student, the course, and the semester.



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 representing the above.



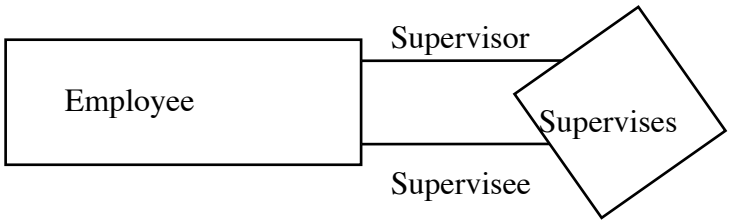
PROJECT

(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.

c. 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

(Note: as drawn we allow an employee to have more than one supervisor, and a supervisor to supervisee more than one employee. If an employee had only one supervisor, we would use an arrow on the Supervisor side of the relationship)

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 CheckOut 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 check outs were as follows (this time using just names and titles as the keys for simplicity):

Borrower	Book
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

PROJECT

This set represents the set of ALL POSSIBLE RELATIONSHIPS that could ever occur; but obviously the CheckOut relationship set 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 cartesian 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.

5. 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. As we have seen, relational DBMS's do not distinguish between entities and relationships; a relationship is, in effect, 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.

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

   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:

   a) 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.

   b) 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.

      Note how this was represented in the notation example for borrowers and books we looked at earlier.

      PROJECT (Notation examples used earlier)

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 CheckOut 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 CheckOut 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 CheckOut is one to many from borrowers to books.

   d. 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, the earlier hierarchical and network models all had 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 relationship as we shall see.

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, the primary key for has ever checked out is the two attributes borrower id and call number together.

Note: Since this is a relationship <u>set</u>, 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. (Or we could include a date attribute in the relationship and make it part of the primary key)

(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 CheckOut is many to one from books to borrowers. Suppose the primary key for books is call number. Then the primary key for CheckOut, like that for books, is call number.

(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 the Chen notation and the notation used in the book 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.

(As an example, in the case of CheckOut, 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.)

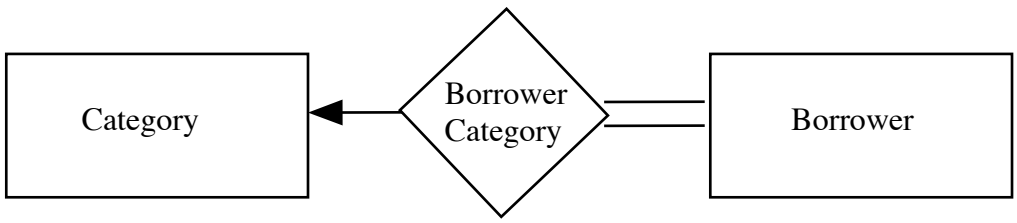
2. Participation constraints: in some cases, the underlying reality dictates that every entity in one of the entity sets must participate in an instance of the relationship.

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

This is represented by using a double line for the connection between the relationship and the entity that must participate.

Example:



PROJECT

(Note: because this diagram uses an arrow head, it also 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. Weak Entities and 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.]

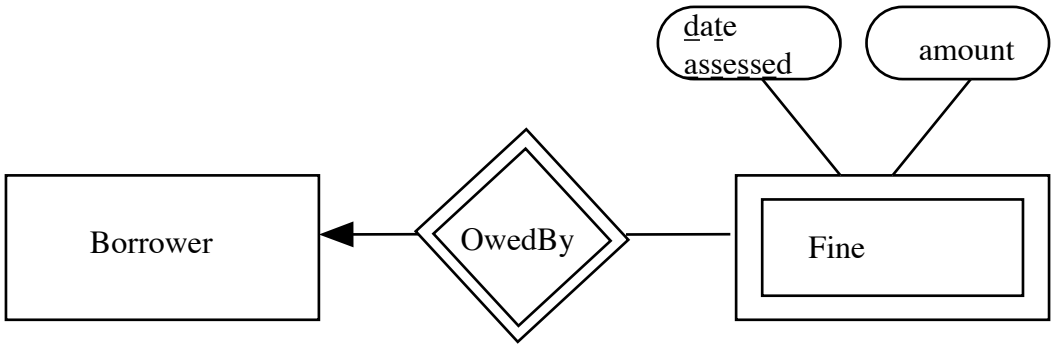
c. Entities within a weak entity set are uniquely identified by the combination of some or all of their own attributes plus the entity on which they depend.

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.Converting an ER diagam to a Relational Schema

A. The book discusses this topic at some length, and we will just hit the high points today.

B. An entity in an ER design can be converted rather straight-forwardly to a relational table. How its attributes are carried over varies with the type of attribute.

1. Simple attributes in the ER-diagram become attributes in the relational schema, typically with the same name and domains.

2. In the case of composite attributes, each individual part becomes a separate attribute attributes in the relational schema, but there is no separate attribute in the relational schema to represent the composite.

3. There are two ways to handle multivalued attributes

   a. A multivalued attribute can be represented by a separate table, with each row holding the primary key of the entity and one of the values. (The primary key serves to connect each row of the table to an entity.

   b. An extension to the relational model (implemented in some way by many commercial RDB's) allows for an attribute to be a collection, with SQL extensions allowing access to individual values.

4. A derived attribute is not represented directly in the relational schema. Instead, code is written to calculate it when needed. Many commercial RDBs allow writing such code in SQL and storing it under its own name in the database.

5. A weak entity can be represented by a table having all the attributes of the entity plus the primary key of the entity its existence depends on.

6. Thus, the borrower entity we looked at previously, together with the weak entity fine, might be converted to a relational schema this way:

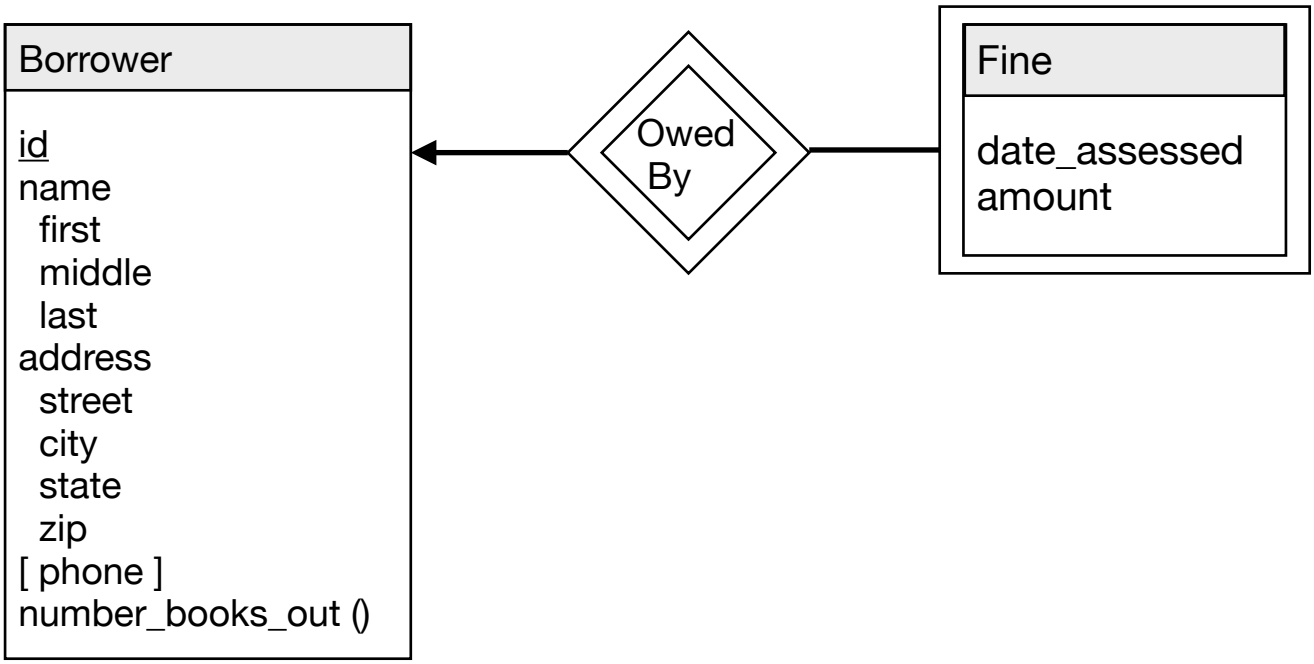
   PROJECT Diagram

   Notes:

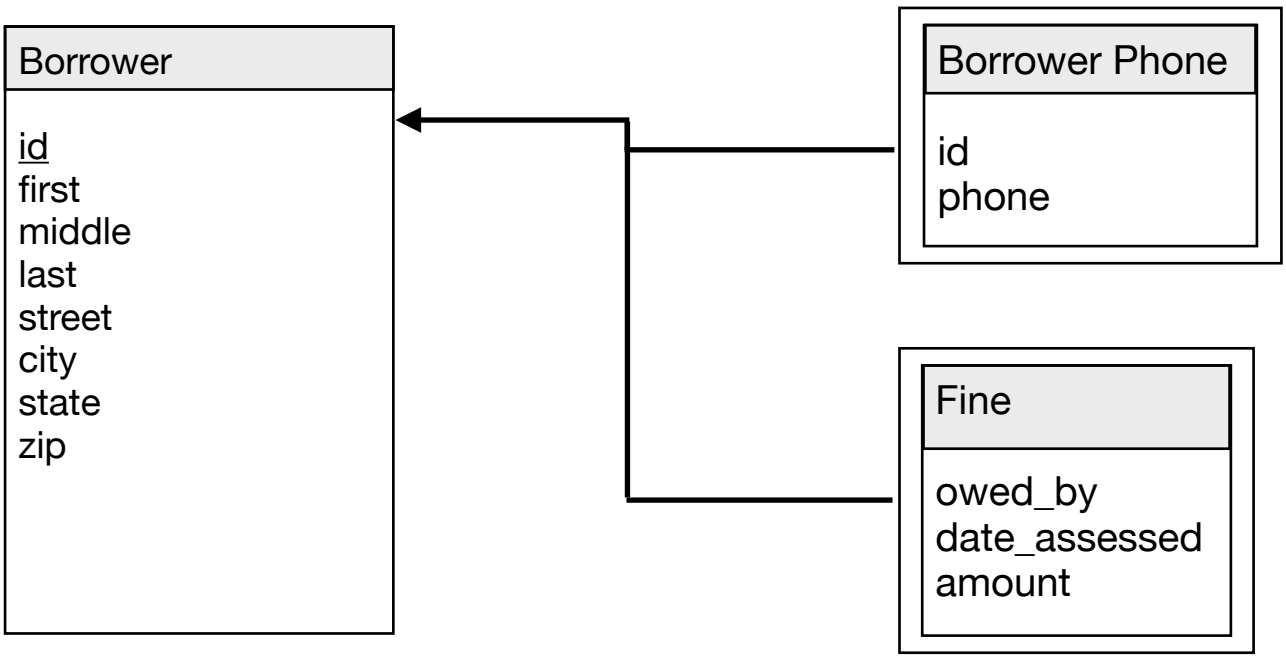
   - The existence dependency for Fines (Owed By) is folded into the corresponding relational table

   - There is also a function number_books_out() that is not explicitly represented in the schema

ER Diagram



Equivalent Relational Schema



C. There are several different says of handling relationships, depending on the multiplicity:

1. A many to many relationship is converted to a separate table, containing the primary keys of the tables being related an any attributes of the relationship.

   The primary key of this table is the union of the primary keys of the tables being related

2. A many to one or one to many relationship can be handled in one of two different ways.

   a. It can be handled as a separate table like the above.

      In this case, though, the primary key is just the primary key of the "one" side of the relationship. since by the nature of the multiplicity constraint it can appear only once in the table.

   b. The primary key of the "many" table and any other attributes of the relationship can be folded into the "one" table. No separate table is needed.

3. A one to one relationship is handled like a many to one or one to many, except that the designer can choose either of the "one" tables to use for the primary key or to fold the relationship into its table."

   One constraint: to avoid needing to use a null value in the selected entity that does not participate in the relationship, it is preferable to choose the entity on the "total" side if the relationship has a total participation constraint.

IV. Tools

Demo www.diagrameditor.com - **important - do not use with Safari - seems to work with Firefox and Chrome. (For demo, open saved file RegistrationExample)**