A Practical Guide to Entity-Relationship Modeling

Il-Yeol Song and Kristin Froehlich
College of Information Science and Technology
Drexel University
Philadelphia, PA 19104

Abstract

The Entity-Relationship (ER) model and its accompanying ER diagrams are widely used for database design and Systems Analysis. Many books and articles just provide a definition of each modeling component and give examples of pre-built ER diagrams. Beginners in data modeling have a great deal of difficulty learning how to approach a given problem, what questions to ask in order to build a model, what rules to use while constructing an ER diagram, and why one diagram is better than another. In this paper, therefore, we present step-by-step guidelines, a set of decision rules proven to be useful in building ER diagrams, and a case study problem with a preferred answer as well as a set of incorrect diagrams for the problem. The guidelines and decision rules have been successfully used in our beginning Database Management Systems course for the last eight years. The case study will provide readers with a detailed approach to the modeling process and a deeper understanding of data modeling.

Introduction

Entity relationship diagrams (ERD) are widely used in database design and systems analysis to represent systems or problem domains. The ERD was introduced by Chen (1976) in early 1976. Teorey, Yang, and Fry (1986) present an extended ER model for relational database design. The ERD models a given problem in terms of its essential elements and the interactions between those elements in a problem domain. The ERD can serve as the basis for databases, which store data about the problem domain, and which use, manipulate, and constrain that data. Experts in systems analysis and database design are adept at identifying user requirements and then translating them into corresponding components of the model. Many books and articles just provide a definition of each modeling component and give examples of pre-built ER diagrams. Beginners in data modeling have a great deal of difficulty learning how to approach a given problem, what questions to ask in order to build a model, what rules to use while constructing an ER diagram, and why one diagram is better than another.

Ahrens and Song (1991) present a set of requirements elicitation template sentences, structured English template sentences, and some decision rules for database modeling. This paper presents a set of heuristic rules which improve upon those presented by Ahrens and Song (1991), together with a detailed case study analysis. We include step-by-step guidelines, a set of decision rules proven to be useful in building ER diagrams, and a case study problem with a preferred answer as well as a set of incorrect diagrams for the problem. These guidelines and decision rules have been successfully used in our beginning Database Management Systems course for the last eight years. The case study will provide readers with a detailed approach to the modeling process and a deeper understanding of data modeling.

The Entity-Relationship Diagram

The entity relationship diagram is a graphical representation of a conceptual structure of a problem domain being modeled. The ERD assists the database designer in identifying the data and the rules that will be represented and used in a database. The ERD is an implementation-independent representation of a problem domain and it facilitates communication between the end-user and the analyst. ERDs can be easily converted into a logical database structure that can be readily implemented in a particular commercial database management system.

The basic components of the ERD are entities, properties of entities called attributes, and relationships between entities.

Entities

Entities are PRIMARY THINGS of a problem domain about which users need to record data. Ross (1988) provides a list of candidate entity types which could be included in the model.

(1) People: humans who carry out some function
    Employees, Students, Customers
(2) Places: sites or locations
    Cities, Offices, Routes
(3) Things: tangible physical objects
    Equipment, Products, Buildings
(4) Organizations
    Teams, Suppliers, Departments
(5) Events: things that happen to some other entity at a given date and time or as steps in an ordered sequence
    Employee promotions, Project phases, Account payments
(6) Concepts: intangible ideas used to keep track of business or other activities
    Projects, Accounts, Complaints

These candidate entity types need to be evaluated against a particular domain being modeled. Some decision rules are discussed in a later section of this paper.

Attributes

Attributes are properties of entities or relationships. Entities have two types of properties: identifying attributes and descriptive attributes. Identifying attributes uniquely determine each instance of an entity type. They are called entity identifiers or *keys*. For example, the attribute *social security number* would uniquely identify each member or instance of the entity type *student*. Descriptive attributes of *student* might include *year, advisor,* and *grade point average*. Each instance of an entity has a value for each attribute. Values for *grade point average* might include *2.5, 3.45,* and *4.0*. Values for year might include *1991*, *1992*, *1993*, and *1994*. Only attributes that are meaningful in terms of modeling the problem under consideration are included in the ERD. For example, we would not include eye color in a student database.

Relationships

Relationships are another basic component of the ERD. A relationship is an association between or among things or entities. A relationship describes a meaningful interaction that needs to be remembered by the system. The degree of a relationship indicates how many entities are participating in the relationship. A *unary* relationship describes an association of an entity with itself. A *binary* relationship, the most common instance, describes an association between two entities. A ternary (or *n-ary*) relationship is an association between three or more entities. The ER methods that allow only unary and binary relationships are called binary models, while ER methods that allow any type of relationship are called n-ary models. For more thorough treatment of ternary relationships, see Jones and Song (1995, 1996) and Song and Jones (1995).

Cardinality and Participation Constraints

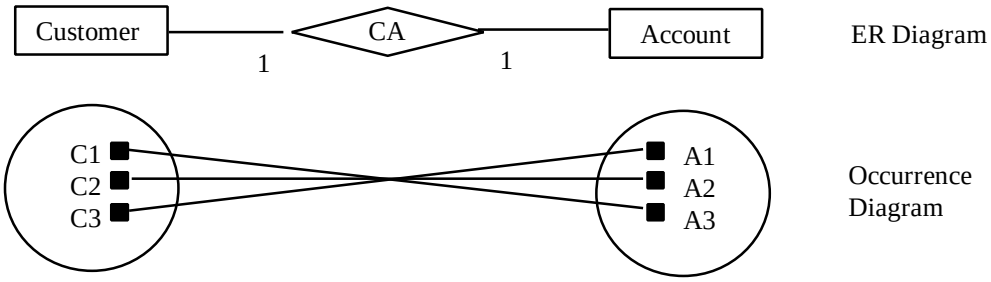
Cardinality is a constraint on the relationship between two entities. Specifically, the cardinality constraint expresses the maximum number of entities that can be associated with another entity via a relationship. For example, in a binary relationship (a relationship with two participating entities), we can have three possible cardinalities: one-to-one (1:1), one-to-many (1:N), or many-to-many (M:N). One-to-one cardinality says that, for entities *customer* and *account*, one *customer* can have at most one *account* and one *account* cannot be owned by more than one *customer*. One-to-many cardinality says that one *customer* can have many *accounts*, but one *account* cannot be owned by more than one *customer*. Many-to-many cardinality says that one *customer* can have many *accounts* and one *account* may be owned by many *customers*.

Participation is also a relationship constraint. Participation expresses the minimum number of entities that can be associated with another entity via a relationship. There are two values for participation: *total* or *mandatory* participation and *partial* or *optional* participation. If every instance of an entity must participate in a given relationship then that entity has total participation in the relationship. But if every instance need not participate in a given relationship then the participation of that entity in the relationship is partial. Given the relationship *employee works for department*, an *employee* has partial participation in that relationship if he or she need not work for a *department*. An *employee* has total participation in the relationship if he or she must work for at least one *department*. Similarly, a *department* has partial participation in the relationship if it can exist without having any *employees*. A *department* has total participation in the relationship if it must have at least one *employee*.

Cardinality and participation constraints are business rules in the problem domain being modeled. These constraints represent the way one entity type is associated with another entity type. These constraints are also integrity constraints because they help to ensure the accuracy of the database. These constraints limit the ways in which data from different parts of the database can be associated. For example, let's say the cardinality of the relationship between *Customer* and *Account* is one-to-one, as in Figure 1(a) below. If customer C1 is associated with account A3, then C1 cannot be associated with any other accounts and A3 cannot be associated with any other customers.

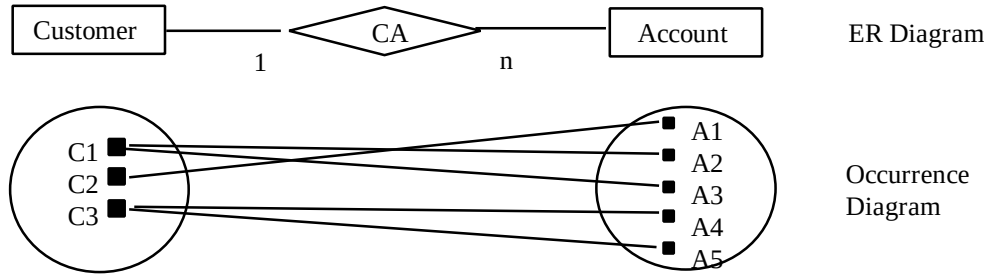
(a) One - to - One (1:1):

One customer can have at most one account.
One account cannot be owned by more than one customer.



(b) One - to - Many (1:n):

One customer can have many accounts.
One account cannot be owned by more than one customer.



(c) Many - to - Many (n:m):

One customer can have many accounts.
One account may be owned by many customers.

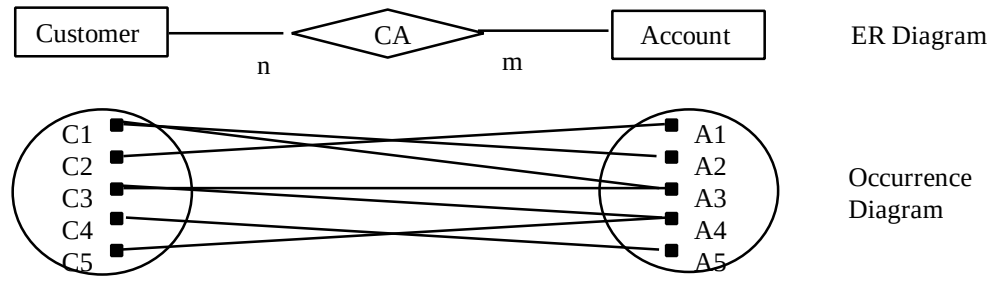


Figure 1. CARDINALITY: The expression of the maximum number of entities that can be associated to another entity via a relationship. Occurrence Diagrams show the relationships between occurrences or instances of each entity.

Taxonomy in ER Modeling

In an ER model, an entity is represented as a *rectangle* containing the name of the entity. The names of attributes are enclosed in an *oval* connected to the rectangle of the entity they describe. Attributes may be omitted from the diagram to avoid cluttering it and also in the early stages of development. Relationships are represented by *diamonds* between entities. The notation of the ERD, however, varies according to the modeling approach used. Binary models do not use the diamond to indicate a relationship, do not represent attributes of relationships, and do not allow ternary relationships, that is, relationships between three or more entities. Martin (1989), Bachman (1992), ERWin and IDEF1X (Bruce, 1992) use the binary modeling approach. Most text books use n-ary modeling, including Elmasri and Navathe (1994), Hawryszkiewycz (1991), Teorey (1994), Batini, Ceri and Navathe (1992), and McFadden and Hoffa (1994). A few notations are illustrated below.

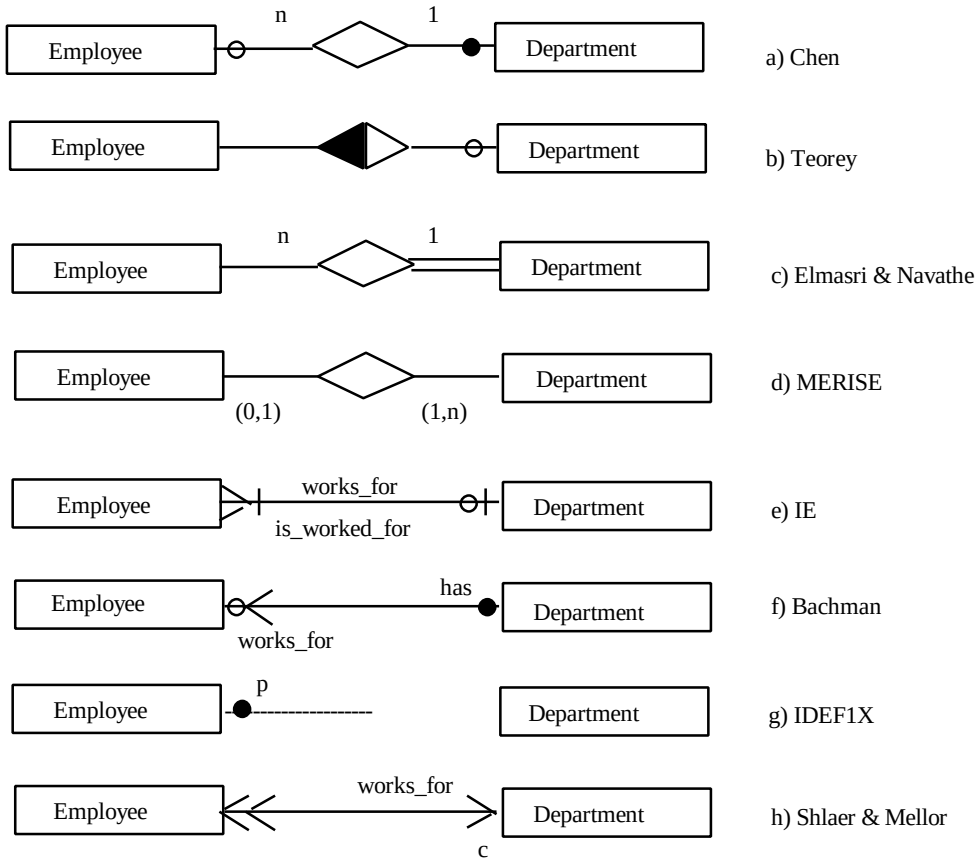


Figure 2. Various notations for ER Diagram representing "one employee works for zero or one department and one department has one or more employees".

Each diagram in Figure 2 contains two entities: *employee* and *department*. In diagrams *a*, *b*, *c*, and *d*, the diamond indicates the relationship between the entities. These diagrams use n-ary modeling. Diagrams *e* through *h* are examples of binary modeling. They do not represent the relationship with the diamond shape. Instead, diagrams *e*, *f*, and *h* label the line between the entities with the relationship name. Attributes were not represented in the diagrams for simplicity. The various circles, lines, arrows, and letters on the diagram indicate cardinality and participation constraints. For a more complete treatment of various ER modeling methods, see Song, Evans, and Park (1995).

ER Modeling

How does one begin creating an entity relationship diagram? In this paper, we present step-by-step guidelines to build an ERD using n-ary modeling using Elmasri and Navathe's notation (see 2.c). In Table 1, we summarize a sequence of steps of database design using an ER model. Note that these steps are iterative.

1. Understand the problem domain. Analyze database requirements.
    - Write a summary specification in English, if not created yet.
    - What do we need to store into the database?
    - What queries and reports do we need to generate?
    - What are *important* people, places, physical things, organizations, events and abstract concepts in the organization?
2. Design a conceptual schema by creating an ER diagram.
    (a) Identify entity types. Assign a singular noun to each entity type.
    (b) Identify relationships between (among) entities. Use a meaningful verb for a relationship name.
    (c) Draw an ERD without attributes.
    (d) Identify relationship cardinalities.
        - Mapping constraint (1:1, 1:N, N:M)
        - Participation constraint (Total, Partial)
    (e) Assign attributes to entity types and relationship types. Usually attributes come from nouns, adjectives or adverbs.
    (f) Select identifiers (primary keys) for entity types.
        - Weak entity: composite primary key.
        - Regular entity: choose/create a single attribute primary key.
    (g) Select the PKs of relationships.
        - If 1:1, then the PK of either side entity type may be selected.
        - If 1:N, then the PK of N-side entity type must be selected.
        - If M:N, then a composite PK consisting of PKs of two entity types must be used.
        - If ternary, then a composite PK consisting of the PKs of at least two entity types. The actual PKs selected will vary depending on the cardinality.
3. Design a logical schema.
    (a) Translate the ERD into a relational schema

- If a relationship cardinality is likely to be changed; use stable method.
- If a relationship cardinality is not likely to be changed; use mapped method.
- If a relationship cardinality is not likely to be changed and null values of foreign keys are significant; use mapped with total/partial method.

(b) Check normalization (at least 3NF).

(c) Create data dictionaries.

- A schema table
- One table for each relation created in step (a)
  - Assign a domain type for each attribute.
  - Explain the meaning of attributes, if not intuitive.
  - Note other values such as range, null, PK, FK, indexed, source, owner

(d) Do database prototyping & modify the design if necessary.

(e) Summarize the design assertion (integrity, security).

4. Verify the design with users. Iterate the steps, if necessary.

Table 1. Steps to DB Design Using ER Modeling

First, it is important to study the problem domain at hand. Analyze database requirements. Write a summary paragraph for the problem domain, considering what data need to be stored and what queries and reports need to be processed. All the information necessary for the identified queries and reports must be included in the summary paragraph. Revise the summary paragraph considering database requirements. Second, from the summary paragraph, find nouns. They are candidates for entity types. To determine whether a noun should be designated as an entity, the following decision rules may be applied.

Rule 1

Every entity type should be important in its own right within the problem domain.

Rule 2

IF an object type (noun) has only one property to store
THEN it is an attribute of another entity type
ELSE it is an entity type.

Rule 3

IF an object type has only one data instance
THEN do not model as an entity type.

Rule 4

IF a relationship needs to have a unique identifier

THEN model it as an entity type.

The first three rules are used to evaluate object types or nouns, and the fourth rule is used to evaluate relationships or verbs.

Example 1

Address is usually a property of another object type, like *customer*, *vendor*, or *company*. Its existence is less important and not meaningful in its own right within the problem domain. Address should be modeled as an attribute.

Example 2

Suppose we are modeling the *customers* of a company and we want to include the *city* where each *customer* resides. If the name of the *city* is its only attribute, then, following Rule 2, model *city* as an attribute not an entity. Similarly, consider the case of modeling *employees* and their *departments.* If the only important property of the *department* is its name, then Rule 2 tells us to model it as an attribute. However, if we need to store additional properties of each *department* such as projects or total sales, then we should consider modeling it as an entity.

Example 3

Consider modeling the activities of a *trucking company*. Since there is only one instance of the *trucking company*, then, according to Rule 3, it is not necessary to represent it in our model as an entity. We note that it is not wrong to model this single instance noun as an entity type. We simply do not model it as an entity type at the conceptual level because it does not add any modeling power.

We need the fourth rule because one fact can be stated in many different ways in English. In the fourth rule, distinguishing between entities and relationships depends on the function the component plays in the problem domain and how data will be stored about it.

Example 4

Consider the three statements *customer orders products, customer pays bills,* and *reviewer reviews papers.* Even though *orders* and *pays* appear to represent relationships, we model them as entities since each instance would need a unique number for identification in real-world situations. Information would be stored in the database for each *order* and *payment*. Each *review* is not likely to need a unique identification number. Instead, we identify each review activity by a combination of Paper# and Reviewer#. Thus, by Rule 4, we model *reviews* as a relationship type.

Once entities have been assigned, we proceed to identify relationship types between those entities. Verbs are useful candidates for relationships. The following question is useful for identifying relationships: "What sentences can be constructed of the form *Entity Verb Entity*?" For example,

- Employee has children (Existence relationship)
- Professor teaches students (Functional relationship)
- Customer places order (Event relationship)

Note that a relationship is not an action of a flow of data as in data flow diagrams. They are important interactions, between two or more entities, that need to be remembered by the system. In the above examples, we want to remember the facts that who is a child of which employee, which professor teaches which students, and which customer places which order. Also keep in mind that all relationships are bi-directional. We should be able to state the relationship in both directions. Expressing the relationships above in the opposite direction yields the following statements:

- Children *belong to* employee
- Students *are taught by* professor
- Order *is placed by* customer

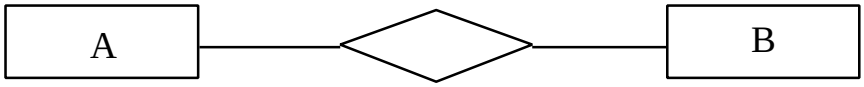
After an ERD has been built, the following rule can aid in validating the diagram.

Rule 5

IF any verb refers to nouns which are not selected as entity types
THEN do not model it as a relationship type.

If any verb in the ERD fails to follow Rule 5, then consider it again carefully before including it in the diagram.

When entities and relationships have been identified, then the cardinality and participation constraints of the relationships can be analyzed. The following rules can help determine the cardinality and participation constraints for a given binary relationship.



Rule 6

For each *A*, what is the maximum number of *Bs* that may be related to it?

Rule 7

IF *A* can exist without being associated with a *B*
THEN *A* has partial (optional) participation
ELSE *A* has total (mandatory) participation.

Example 5

Consider the relationship *Supplier Supplies Account*. For each *Supplier*, what is the maximum number of *Accounts* that may be related to it? Let's say that in our problem domain, each *Supplier* may have many *Accounts* but each *Account* may have only one *Supplier*. By Rule 6, the cardinality constraint for Supplier:Account is 1:N or one to many. Figure 1 illustrates the cardinality constraints.

Example 6

In determining the participation constraint of *Supplier Supplies Account*, we follow Rule 7: If *Supplier* can exist without being associated with *Account*, THEN *Supplier* has partial participation, ELSE *Supplier* has total participation. In our problem domain, *Supplier* may exist without being associated with *Account*. Therefore, *Supplier* has partial participation in the *Supply* relationship. However, since *Account* cannot exist without a *Supplier*; *Account* has total participation in the *Supply* relationship.

Some basic naming conventions have been established to maintain accuracy and consistency in the database and to avoid redundancy. All entity names should be unique. Use singular nouns in the diagram for both entity and attribute names. Use verbs in the present tense for relationship names. Verbs should be meaningful. For example, avoid verbs like *is*, *has*, and *do* whenever possible. Additionally, well-defined ERDs should satisfy the following basic rules:

- All entities and relationships should be connected.
- All entity names should be unique.
- Each entity must have at least one relationship.
- A relationship cannot be directly connected to another relationship.
- Every entity must have at least one unique attribute, which serves to identify each instance of that entity.

Case Study

The following example will illustrate our guidelines for modeling requirements of the problem domain with entity-relationship diagrams. Using the summary paragraph of the problem description below, we will progress through the steps described above. The nouns in the problem description appear in boldface and the verbs are italicized to aid in the following analysis.

Summary Paragraph of Problem Description

A database specialist wants to design a part of the database for a small drug store owner as follows:

The **owner** wants to *keep track* of all the **suppliers** who *supply* anything to the store. For each supplier, the owner *assigns* a unique **supplier number**, and wants to *keep* the

company name, **address** (**number, street, city, state, zip**), **contact person's name**, **phone number**, **fax number**, and a **comment** for each supplier. For each supply activity, an **account** is *established* to *keep track* of the **date incurred**, the **total cost** of the activity, **due date** for payment, **outstanding balance** after some payments, and any special **comments** related to the account. For each account, the owner may *pay* at several different times and in different ways (e.g., **cash, check, credit card**). For each **payment** activity, the owner wants to *keep* the **date of payment**, **amount of payment**, **method of payment** (check: **check number**; credit card: **credit card name**, **type**, and **number**). Note that one supplier can supply many times and one payment can pay for several accounts of the same supplier.

Entity Analysis

After reading and understanding the problem statement, our first step is to identify entities for the ERD. To do that we examine the nouns in the problem statement. Nouns appear in boldface. We test each noun against our four criteria to determine whether or not it should be included as an entity type. Our first noun is *owner*. Recall that an entity type has more than one instance and more than one property. Since there is only one instance of *owner*, we do not model it as an entity type. Similarly, there is only one *store*, so we need not represent *store* as an entity type.

The next noun, *supplier*, can be classified as an entity type. Several properties of *supplier* are listed in the problem statement. The statement also refers to more than one *supplier*. Therefore, according to Rules 1, 2, and 3, we model *supplier* as an entity. For each *supplier*, the owner wants to store the following properties in the database: *supplier number*, *company name*, *contact person*, *address*, *phone number*, *fax number*, and *comment*. Each of these attributes except *address* has only one property to store so we model them as attributes. *Address* has its component properties number, city, state, and zip so one might be tempted to model it as an entity type. However, the role of *address* as a property of *supplier* supersedes the fact that *address* has properties of its own. In other words, *address* itself without *supplier* is not important in its own right. Therefore, by Rule 1, we model *address* as an attribute.

Account is the next noun. *Account* has several properties to be stored in the database: *date incurred*, *total cost*, *due date*, *account balance*, and *comments*; and we will store information about numerous *accounts*. Therefore, we designate *account* as an entity. Its properties are modeled as attributes of *account.*

Payment is clearly an entity, with multiple instances and various properties. The properties of *payment*: *date of payment*, *amount of payment*, and *method of payment*, are modeled as its attributes. *Cash*, *check*, and *credit card* appear to be attributes of *payment*, but actually, they are not attributes themselves, but simply different values for the attribute *method of payment*. This distinction becomes clearer if we think about

storing data in the database. For each *payment*, one of the values *cash*, *check*, or *credit card* will be stored in the location containing data about the *method of payment*. *Check number* and *credit card name*, *type* and *number* may be modeled as attributes of Payment.

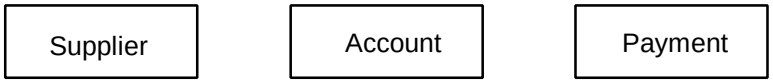


Figure 3. Entities to be included in the ERD.

Relationship Analysis

Our analysis of nouns in the problem statement has produced three entities: *Supplier*, *Account*, and *Payment* (Figure 3). Keep these entities in mind as we identify relationships between them. Let's examine the verbs in the problem statement as candidates for relationships in the diagram. Verbs appear in italics. Of the verbs in the problem statement: *keep track*, *assigns*, *supply*, *established*, and *pay*, only *supply* and *pay* are possible candidates for relationships between the entities *account*, *supplier*, and *payment*. *Keep track* and *keep* appear several times in the problem statement. These terms refer, not to a relationship between entities, but generally to storing data in the database. In other words, they are used to describe the problem domain, not an interaction that needs to be remembered by the system. Therefore, we do not model them as relationships.

Established, in the statement an *account is established*, is an activity performed by the owner or the system itself. Similarly, *owner assigns a unique supplier number* reflects an activity by the owner. These two verbs do not represent relationships between any of our three entities. Thus, we are left with the verbs *supply* and *pay*.

A *supplier* performs a supply activity. The result of a supply activity is an *account*. Therefore, a good candidate for the relationship between *supplier* and *account* is *supply*. Stated in both directions, the relationship is *Supplier supplies account* and *account is supplied by supplier*. Rule 4 states that if a relationship needs to have a unique identifier, then model it as an entity. Each supply activity is unique, so we may be tempted to model *supply* as an entity. However, the data for each activity is stored using the entity *account*, so it is not necessary to create another entity which stores the same information.

Each *payment* credits an *account* so *pay* is the relationship between *payment* and *account*. Expressing the relationship *pay* in both directions, we can say *account is paid by payment* and *payment pays account*.

Now we can draw the basic ERD (Figure 4). We include the entities *Supplier*, *Account*, and *Payment*, and the relationships *Supply* and *Pay*. Attributes may be added to the diagram at this point or omitted to avoid clutter.

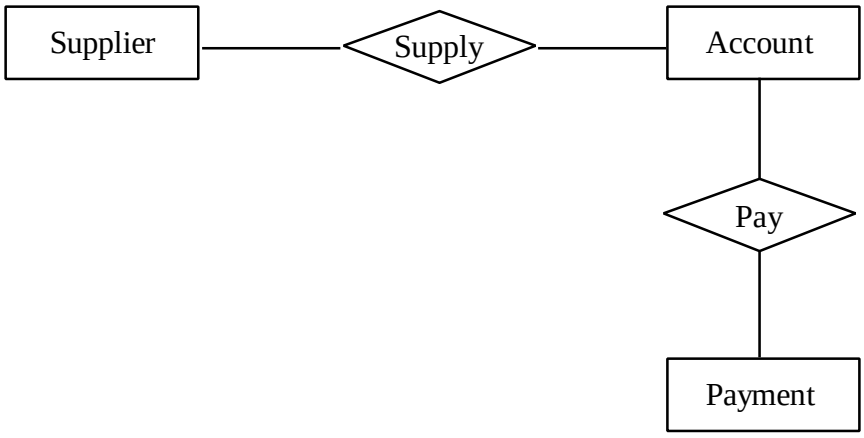


Figure 4. ERD without attributes and constraints

Analysis of Cardinality and Participation Constraints

In order to identify the cardinality and participation constraints of each relationship in the ERD, we follow Rules 6 and 7 looking at the relationship first from the point of view of one entity and then from the other entity. In our ERD above, to determine the cardinality constraint of the relationship *Supply*, we begin by asking, "For each *Supplier*, what is the maximum number of *Accounts* that may be created?" From the problem statement, we know that one *supplier* can *supply* many times and an *account* is established for each supply activity. Viewing the relationship in the other direction, we ask, "What is the maximum number of *Suppliers* for which each *Account* may contain information?" From the problem statement we can assume that each *account* carries information for a single *supplier*, since *accounts* are established for individual supply activities. Thus, for each *supplier*, there may be many *accounts* and each *account* may have only one *supplier*. The relationship *Supplier Supplies Account* is a one-to-many relationship. The diagram is marked with a *1* on the side of the relationship *Supply* nearer to *Supplier*, and an *N* (for many) on the side nearer to *Account* (see Figure 5).

To identify the cardinality of the relationship *Payment Pays Account*, we look at the relationship from both directions. We ask, "What is the maximum number of *Payments* we can accept for each *Account*?" The answer is clearly stated in the problem statement: For each *account*, the owner may pay at several different times and in different ways. From the opposite direction, "For each *Payment*, what is the maximum number of *Accounts* for which it may pay?" Again, we find the answer in the problem statement: One *payment* can pay for several *accounts* of the same *supplier*. In sum, each *account*

may receive many *payments* and each *payment* may pay for many *accounts*. Therefore, the relationship *Payment Pays Account* is many-to-many. This time, we mark our diagram with an *M* on one side of the relationship *Pay* and an *N* on the other side. (Note that the use of *M* or *N* is completely arbitrary.)

We go through a similar process to determine the participation constraint of each relationship, looking at the relationship from each direction. For the *Supply* relationship we ask, "Can a *Supplier* exist without generating *Accounts*?" In the other direction, "Can an *Account* exist without having *Suppliers supply* merchandise?" The answers to these questions are not explicit in the problem statement. In a real world situation, the database designers would clarify questions like these with the owner. In this case, we will make assumptions from what we understand about the problem domain. *Suppliers* are generally fairly stable entities. A company maintains relationships with several regular *suppliers* regardless of whether they have outstanding *accounts*. On the other hand, an *account* is only created when a *supplier* supplies merchandise. Since *suppliers* can exist without having current *accounts*, *Supplier* has partial participation in the *Supply* relationship. *Accounts*, however, depend on *suppliers* for their existence. Thus, *Account* has total participation in the *Supply* relationship.

To determine the participation of the entities *Payment* and *Account* in the *Pay* relationship, we ask, "Can a *Payment* exist without paying for an *Account* ?" and "Can an *Account* exist without receiving *Payments* against it?" A *payment* which pays for nothing is absurd. It cannot exist without an *account*. An *account*, however, may exist without receiving *payments* against it. Therefore, *Payment* has total participation and *Account* has partial participation in the relationship *Pay*.

In representing the cardinality and participation constraints described above in our ERD, we will employ Elmasri and Navathe's (1994) notation. If an entity has partial participation in the relationship, then a single line is drawn on the line between that entity and the relationship. A double line indicates total participation. The cardinality constraint is represented by Look Across convention, while participation constraint is represented by Look Here convention. Figure 5 illustrates the final ERD with cardinality and participation constraints.

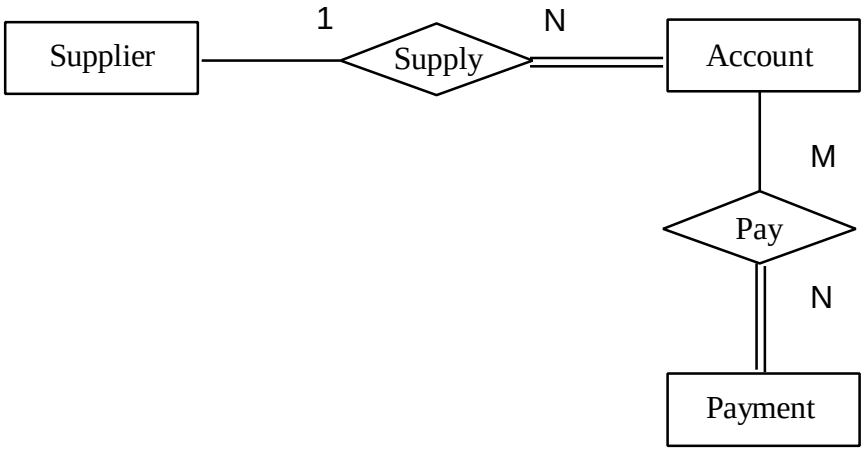


Figure 5. ERD with cardinality and participation constraints.

Errors in Modeling

A common error that novice designers make is failing to recognize the boundaries of a problem domain. They fail to make a distinction between elements that comprise the content of the database and elements that are outside the scope of the database. For example, in the problem statement above, a novice might want to model the verbs *keep track* or *assigns* or *established* as relationships (see Figure 6(a)). These verbs refer to implementing the database and not to its content. *Keep track* refers to storing data in the database, *established* refers to adding an instance of an entity to the database, and *assigns* refers to giving a value to an attribute of an entity. In deciding which elements to model, it is valuable to keep in mind the real world situation.

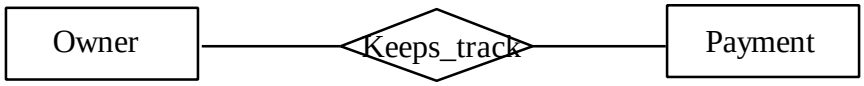
Novice designers also frequently confuse entities with their attributes or properties, as in Figure 6(b). Occasionally, if properties are complex and play a significant role in the problem domain, then they may be modeled as entities. More often, however, properties of an entity should be modeled as attributes. In our problem statement, a novice user may decide to model *address*, a property of the entity *supplier*, as an entity. Modeling *Address* follows Rules 2 and 3 about identifying entities: it has more than one property and it has more than one occurrence. However, *address* does not follow Rule1 in that it is not important in its own right. The role of *address* in the database is more accurate as an attribute of *supplier*, than as an entity with its own relationships.

Other errors are modeling indirect or redundant relationships and inappropriately modeling object types as relationships rather than as entities. Given our problem statement, one may be tempted to model the relationship *Payment Pays Supplier* as in Figure 6(c) or *Supplier Pays Account* as in Figure 6(d) rather than *Payment Pays Account*. Figure 6(c) represents the association between *payment* and *account* indirectly. This indirect relationship can only exist after we have all the direct

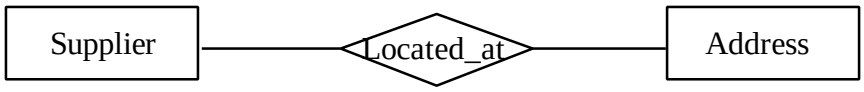
relationships as in Figure 5. In this case, the indirect relationship simply becomes redundant. Without the direct relationships, the indirect relationship cannot be added, because it cannot explain how a particular payment is distributed to multiple accounts. Figure 6(d) represents the relationship *Pay* rather than the entity *Payment*. In either of these two cases, it is difficult to explicitly represent the fact that one *payment* can *pay* for several *accounts* of the same *supplier*. We can only tell implicitly by reading the check number for the various *payments*. If the *payment* is made in cash, there is no way to identify that it paid for more than one *account*.

If the representation used in Figure 6(d) is used, then the attributes related to *payments*: *date of payment*, *amount of payment*, and *method of payment*, are now attributes of the relationship *Pay*. This representation can add unnecessary complexity to the model. Ordinarily, a relationship is uniquely represented by the identifiers of one or more of the entities which participate in it. If the relationship includes a time-dependent attribute like *date of payment*, then that attribute must also be included in the primary key for that relationship. Additionally, instances of *date of payment* and *amount of payment* will require redundant representation because they will have to be included for each *account* covered by a *payment*. Finally, in business practice, each payment activity usually requires a unique identifier. Therefore, following Rule 4, it is more appropriate to model *payment* as an entity than as a relationship. As an entity, the representation is more straightforward and less likely to include redundant or inaccurate information.

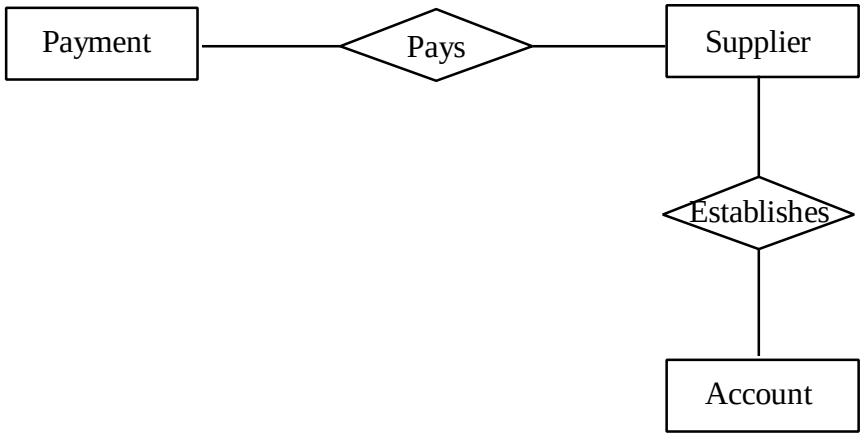
(a) Selection of wrong verb as relationship



(b) Attribute as entity



(c) Indirect relationship



(d) Payment as relationship instead of entity

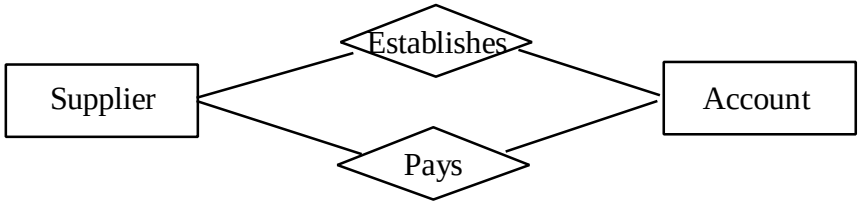


Figure 6. Errors in ERD Modeling.

Limitations of Guidelines and Rules

Two limitations of our guidelines are that they don't account for incomplete requirements analysis or for ambiguity in the problem description. If the problem description is incomplete, then the resulting analysis based on this approach will also be incomplete. We assume that the analysis is complete. If the problem specification is modified, the analysis and resulting ERD should be modified as well. In English, one concept can be represented in many different ways. For example, we can say that *customer orders products* or *customer places an order to buy products. Order* is used

as a verb in the first sentence and as a noun in the second. We minimize this problem by adopting Rule 4, which states that if a verb needs to have a unique identifier, we model it as an entity type rather than a relationship type.

Conclusion

We have discussed a set of decision rules which are useful in building ERDs and have illustrated the application of these rules using a single example. ERD constructs discussed here include Entities, Relationships, Attributes, Cardinality constraints and Participation constraints. To simplify our discussion, we didn't include other constructs such as Weak Entity, Ternary Relationship, and Generalization/ Specialization. Our rules are heuristics which we have found useful for most cases to build ERDs in the early stages of analysis. However, these rules may need some refinement in some problem domains and the rules should be adapted to the problem domain under consideration.

References

Ahrens, J. and Song, I.Y. (1991). "EER Data Modeling Aids for Novice Database Designers". *Proceedings of the 2nd International Conferences of the Information Resources Management*, Memphis, TN, May 19-22, 1991, pp. 99-114.

Bachman (1992). Bachman Analyst, Bachman Information Systems Incorporated.

Batini, C., Ceri, S., and Navathe, S. (1992). *Conceptual Database Design: An Entity-Relationship Approach*, Redwood City, CA: Benjamin/Cummings Publishing Company, Inc.

Bruce, T. (1992). *Designing Quality Databases with IDEF1X Information Models*. New York, New York: Dorset House Publishing.

Chen, P.P. (1976). "The Entity Relationship Model - Toward a Unified View of Data". *ACM Transactions on Database Systems*, 1 :1, pp. 9-36.

Elmasri, R. and Navathe, S. (1994). *Fundamentals of Database Systems*, 2nd ed., Redwood City, CA: Benjamin/ Cummings Publishing Company, Inc.

Hawryszkiewycz, I.T. (1991). *Database Analysis and Design*, 2nd ed., MacMillan Publishing Company.

Jones, T.H. and Song, I.-Y., (1995). "Binary Representation of Ternary Relationships in ER Conceptual Modeling," in *14th Int'l Conf. on Object-Oriented and Entity-Relationship Approach*, December 12-15, 1995, Australia, pp. 216-225. (*Object-*

Oriented and Entity-Relationship Approach, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1021).

Jones, T.H. and Song, I.-Y., (1996). "Analysis of Binary/ternary Cardinality Combinations in Entity-Relationship Modeling," *Data & Knowledge Engineering* Vol 19, No. 1, pp. 39-64.

Martin, J. (1989). *Information Engineering: Book II: Planning and Analysis*, Englewood Cliffs, NJ: Prentice Hall.

McFadden, F., and Hoffa, J. (1994). *Modern Database Management*, 4th Ed., Redwood City, CA: Benjamin/Cummings Publishing Company, Inc.

Ross, R.G. (1988). *Entity Modeling: Techniques and Application*, Database Research Group, Inc.

Shaler, S. and Mellor, S.J. (1988). *Object-Oriented Systems Analysis: Modeling the World in Data*, Englewood Cliffs, NJ: Yourdon Press.

Song, I-Y., Evans, M., and Park, E.K. (1995). "A Comparative Analysis of Entity-Relationship Diagrams," *Journal of Computer and Software Engineering*, Vol. 3, No.4 (1995), pp. 427-459.

Song, I.Y. and Jones, T.H. (1995). "Ternary Relationship Decomposition Strategies Based on Binary Imposition Rules," in *11th Int'l Conf. on Data Engineering*, March 6-10, 1995, Taipei, Taiwan, pp. 485-492.

Teorey, T.J. (1994). *Database Modeling & Design: The Fundamental Principles*, 2nd. ed., Morgan Kauffman Publishers, Inc.

Teorey, T.J., Yang, D., and Fry, J.P., (1986). "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model". *Computing Surveys*, 18:12, June, pp. 197-222.