**CHAPTER THREE**

**DATABASE MODELS**

**Introduction**

A data model is a conceptual representation of the data structures that are required by a database. The data structures include the data objects, the associations between data objects, and the rules which govern operations on the objects. As the name implies, the data model focuses on what data is required and how it should be organized rather than what operations will be performed on the data. To use a common analogy, the data model is equivalent to an architect's building plans.

A data model is independent of hardware or software constraints. Rather than try to represent the data as a database would see it, the data model focuses on representing the data as the user sees it in the "real world". It serves as a bridge between the concepts that make up real-world events and processes and the physical representation of those concepts in a database.

**Methodology**

There are two major methodologies used to create a data model: the Entity-Relationship (ER) approach and the Object Model. This document uses the Entity-Relationship approach.

**Data Modeling In the Context of Database Design**

Database design is defined as: "design the logical and physical structure of one or more databases to accommodate the information needs of the users in an organization for a defined set of applications". The design process roughly follows five steps:

1. Planning and analysis

2. Conceptual design

3. Logical design

4. Physical design

5. Implementation

The data model is one part of the conceptual design process. The other, typically is the functional model. The data model focuses on what data should be stored in the database while the functional model deals with how the data is processed. To put this in the context of the relational database, the data model is used to design the relational tables. The functional model is used to design the queries which will access and perform operations on those tables.

**Components of a Data Model**

The data model gets its inputs from the planning and analysis stage. Here the modeler, along with analysts, collects information about the requirements of the database by reviewing existing documentation and interviewing end-users.

The data model has two outputs. The first is an entity-relationship diagram which represents the data structures in a pictorial form. Because the diagram is easily learned, it is valuable tool to communicate the model to the end-user. The second component is a data document. This a document that describes in detail the data objects, relationships, and rules required by the database. The dictionary provides the detail required by the database developer to construct the physical database.

**Why is Data Modeling Important?**

Data modeling is probably the most labor intensive and time consuming part of the development process. Why bother especially if you are pressed for time? A common response by practitioners who write on the subject is that you should no more build a database without a model than you should build a house without blueprints.

The goal of the data model is to make sure that the all data objects required by the database are completely and accurately represented. Because the data model uses easily understood notations and natural language , it can be reviewed and verified as correct by the end-users.

The data model is also detailed enough to be used by the database developers to use as a "blueprint" for building the physical database. The information contained in the data model will be used to define the relational tables, primary and foreign keys, stored procedures, and triggers. A poorly designed database will require more time in the long- term. Without careful planning you may create a database that omits data required to create critical reports, produces results that are incorrect or inconsistent, and is unable to accommodate changes in the user's requirements.

**The Entity-Relationship Model**

The Entity-Relationship (ER) model was originally proposed by Peter in 1976 as a way to unify the network and relational database views. Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually

represents data objects. Since Chen wrote his paper the model has been extended and today it is commonly used for database design For the database designer, the utility of the ER model is:

It maps well to the relational model. The constructs used in the ER model can easily be transformed into relational tables.

It is simple and easy to understand with a minimum of training. Therefore, the model can be used by the database designer to communicate the design to the end user.

In addition, the model can be used as a design plan by the database developer to implement a data model in a specific database management software.

**Basic Constructs of E-R Modeling**

The ER model views the real world as a construct of entities and association between entities.

**Entities**

Entities are the principal data object about which information is to be collected. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database. Some specific examples of entities are EMPLOYEES, PROJECTS, INVOICES. An entity is analogous to a table in the relational model.

Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An independent entity is one that does not rely on another for identification. A dependent entity is one that relies on another for identification.

An entity occurrence (also called an instance) is an individual occurrence of an entity. An occurrence is analogous to a row in the relational table.

**Special Entity Types**

Associative entities (also known as intersection entities) are entities used to associate two or more entities in order to reconcile a many-to-many relationship.

Subtypes entities are used in generalization hierarchies to represent a subset of instances of their parent entity, called the supertype, but which have attributes or relationships that apply only to the subset.

Associative entities and generalization hierarchies are discussed in more detail below.

**Relationships**

A Relationship represents an association between two or more entities. An example of a relationship would be:

Employees are assigned to projects

Projects have subtasks

Departments manage one or more projects

Relationships are classified in terms of degree, connectivity, cardinality, and existence. These concepts will be discussed below.

**Attributes**

Attributes describe the entity of which they are associated. A particular instance of an attribute is a value. For example, "Jane R. Hathaway" is one value of the attribute Name. The domain of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string.

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non- unique characteristic of an entity instance.

**Classifying Relationships**

Relationships are classified by their degree, connectivity, cardinality, direction, type, and existence. Not all modeling methodologies use all these classifications.

**Degree of a Relationship**

The degree of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary, and ternary ,where the degree is 2, and 3, respectively.

Binary relationships, the association between two entities is the most common type in the real world. A recursive binary relationship occurs when an entity is related to itself. An example might be "some employees are married to other employees".

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decomposed into two or more binary relationships.

**Connectivity and Cardinality** The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or

"many". The cardinality of a relationship is the actual number of related occurences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one- to-many, and many-to-many.

A *one-to-one* (1:1) relationship is when at most one instance of a entity A is associated with one instance of entity B. For example, "employees in the company are each assigned their own office. For each employee there exists a unique office and for each office there exists a unique employee.

A *one-to-many* (1:N) relationships is when for one instance of entity A, there are zero, one, or many instances of entity B, but for one instance of entity B, there is only one instance of entity A. An example of a 1:N relationships is

A department has many employees

Each employee is assigned to one department

A *many-to-many* (M:N) relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A. An example is: employees can be assigned to no more than two projects at the same time;

projects must have assigned at least three employees

A single employee can be assigned to many projects; conversely, a single project can have assigned to it many employee. Here the cardinality for the relationship between employees and projects is two and the cardinality between project and employee is three. Many-to-many relationships cannot be directly translated to relational tables but instead must be transformed into two or more one-to-many relationships using associative entities.

**Direction**

The direction of a relationship indicates the originating entity of a binary relationship. The entity from which a relationship originates is the parent entity; the entity where the relationship terminates is the child entity.

The direction of a relationship is determined by its connectivity. In a one-to-one relationship the direction is from the independent entity to a dependent entity. If both entities are independent, the direction is arbitrary. With one-to-many relationships, the

entity occurring once is the parent. The direction of many-to-many relationships is arbitrary.

**Type**

An *identifying relationship* is one in which one of the child entities is also a dependent entity. A *non-identifying relationship* is one in which both entities are independent. **Existence**

*Existence* denotes whether the existence of an entity instance is dependent upon the existence of another, related, entity instance. The existence of an entity in a relationship is defined as either *mandatory* or *optional*. If an instance of an entity must always occur for an entity to be included in a relationship, then it is mandatory. An example of mandatory existence is the statement "every project must be managed by a single department". If the instance of the entity is not required, it is optional. An example of optional existence is the statement, "employees may be assigned to work on projects". **Generalization Hierarchies**

A generalization hierarchy is a form of abstraction that specifies that two or more entities that share common attributes can be generalized into a higher level entity type called a *supertype* or *generic entity*. The lower-level of entities become the subtype, or categories, to the supertype. Subtypes are dependent entities.

Generalization occurs when two or more entities represent categories of the same real- world object. For example, Wages\_Employees and Classified\_Employees represent categories of the same entity, Employees. In this example, Employees would be the supertype; Wages\_Employees and Classified\_Employees would be the subtypes.

Subtypes can be either mutually exclusive (disjoint) or overlapping (inclusive). A mutually exclusive category is when an entity instance can be in only one category. The above example is a mutually exclusive category. An employee can either be wages or classified but not both. An overlapping category is when an entity instance may be in two or more subtypes. An example would be a person who works for a university could also be a student at that same university. The completeness constraint requires that all instances of the subtype be represented in the supertype. Generalization hierarchies can be nested. That is, a subtype of one hierarchy can be a supertype of another. The level of

nesting is limited only by the constraint of simplicity. Subtype entities may be the parent entity in a relationship but not the child.

**ER Notation**

There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation. All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of a connection. The notation used in this document is from Martin. The symbols used for the basic ER constructs are:

Entities are represented by labeled rectangles. The label is the name of the entity.

Entity names should be singular nouns.

Relationships are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs.

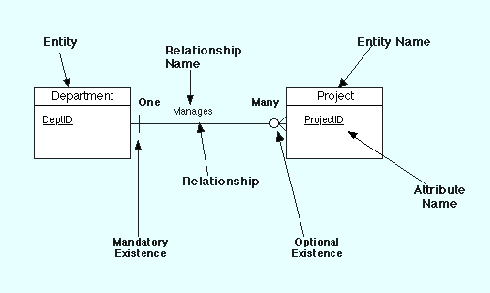
Attributes, when included, are listed inside the entity rectangle. Attributes which are identifiers are underlined. Attribute names should be singular nouns.

Cardinality of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.

Existence is represented by placing a circle or a perpendicular bar on the line.

Mandatory existence is shown by the bar (looks like a 1) next to the entity for an instance is required. Optional existence is shown by placing a circle next to the entity that is optional.

Examples of these symbols are shown in Figure 2.1 below:



**ER Notation**

**Data Modeling As Part of Database Design**

The data model is one part of the conceptual design process. The other is the **function model.** The data model focuses on what data should be stored in the database while the function model deals with how the data is processed. To put this in the context of the relational database, the data model is used to design the relational tables. The functional model is used to design the queries that will access and perform operations on those tables.

Data modeling is preceeded by planning and analysis. The effort devoted to this stage is proportional to the scope of the database. The planning and analysis of a database intended to serve the needs of an enterprise will require more effort than one intended to serve a small workgroup.

The information needed to build a data model is gathered during the requirments analysis. Although not formally considered part of the data modeling stage by some methodologies, in reality the requirements analysis and the ER diagramming part of the data model are done at the same time.

**Requirements Analysis**

The goals of the requirements analysis are:

To determine the data requirements of the database in terms of primitive objects

To classify and describe the information about these objects

To identify and classify the relationships among the objects

To determine the types of transactions that will be executed on the database and the interactions between the data and the transactions

To identify rules governing the integrity of the data

The modeler, or modelers, works with the end users of an organization to determine the data requirements of the database. Information needed for the requirements analysis can be gathered in several ways:

**Review of existing documents** - such documents include existing forms and reports, written guidelines, job descriptions, personal narratives, and memoranda. Paper documentation is a good way to become familiar with the organization or activity you need to model.

**Interviews with end users** - these can be a combination of individual or group meetings. Try to keep group sessions to under five or six people. If possible, try to have everyone with the same function in one meeting. Use a blackboard, flip charts, or overhead transparencies to record information gathered from the interviews.

**Review of existing automated systems** - if the organization already has an automated system, review the system design specifications and documentation

The requirements analysis is usually done at the same time as the data modeling. As information is collected, data objects are identified and classified as either entities, attributes, or relationship; assigned names; and, defined using terms familiar to the end- users. The objects are then modeled and analysed using an ER diagram. The diagram can be reviewed by the modeler and the end-users to determine its completeness and accuracy. If the model is not correct, it is modified, which sometimes requires additional information to be collected. The review and edit cycle continues until the model is certified as correct.

Three points to keep in mind during the requirements analysis are:

1. Talk to the end users about their data in "real-world" terms. Users do not think in terms of entities, attributes, and relationships but about the actual people, things, and activities they deal with daily.

2. Take the time to learn the basics about the organization and its activities that you want to model. Having an understanding about the processes will make it easier to build the model.

3. End-users typically think about and view data in different ways according to their function within an organization. Therefore, it is important to interview the largest number of people that time permits.

**Steps In Building the Data Model**

While ER model lists and defines the constructs required to build a data model, there is no standard process for doing so. Some methodologies, such as IDEFIX, specify a bottom-up development process were the model is built in stages. Typically, the entities and relationships are modeled first, followed by key attributes, and then the model is finished by adding non-key attributes. Other experts argue that in practice, using a phased

approach is impractical because it requires too many meetings with the end-users. The sequence used for this document are:

1. Identification of data objects and relationships

2. Drafting the initial ER diagram with entities and relationships

3. Refining the ER diagram

4. Add key attributes to the diagram

5. Adding non-key attributes

6. Diagramming Generalization Hierarchies

7. Validating the model through normalization

8. Adding business and integrity rules to the Model

In practice, model building is not a strict linear process. As noted above, the requirements analysis and the draft of the initial ER diagram often occur simultaneously. Refining and validating the diagram may uncover problems or missing information which require more information gathering and analysis

**Identifying Data Objects and Relationships**

In order to begin constructing the basic model, the modeler must analyze the information gathered during the requirements analysis for the purpose of:

Classifying data objects as either entities or attributes

Identifying and defining relationships between entities

Naming and defining identified entities, attributes, and relationships

Documenting this information in the data document

To accomplish these goals the modeler must analyze narratives from users, notes from meeting, policy and procedure documents, and, if lucky, design documents from the current information system.

Although it is easy to define the basic constructs of the ER model, it is not an easy task to distinguish their roles in building the data model. What makes an object an entity or attribute? For example, given the statement "employees work on projects". Should employees be classified as an entity or attribute? Very often, the correct answer depends upon the requirements of the database. In some cases, employee would be an entity, in some it would be an attribute.

While the definitions of the constructs in the ER Model are simple, the model does not address the fundamental issue of how to identify them. Some commonly given guidelines are:

Entities contain descriptive information

Attributes either identify or describe entities

Relationships are associations between entities

These guidelines are discussed in more detail below.

Entities

Attributes

o Validating Attributes

o Derived Attributes and Code Values

Relationships

Naming Data Objects

Object Definition

Recording Information in Design Document

**Entities**

There are various definitions of an entity:

"Any distinguishable person, place, thing, event, or concept, about which information is kept"

"A thing which can be distinctly identified"

"Any distinguishable object that is to be represented in a database"

"...anything about which we store information (e.g. supplier, machine tool, employee, utility pole, airline seat, etc.). For each entity type, certain attributes are stored".

These definitions contain common themes about entities:

o An entity is a "thing", "concept" or, object". However, entities can sometimes represent the relationships between two or more objects. This type of entity is known as an associative entity.

o Entities are objects which contain descriptive information. If an data object you have identified is described by other objects, then it is an entity. If there is no descriptive information associated with the item, it is not an entity. Whether or

not a data object is an entity may depend upon the organization or activity being modeled.

o An entity represents many things which share properties. They are not single things. For example, King Lear and Hamlet are both plays which share common attributes such as name, author, and cast of characters. The entity describing these things would be PLAY, with King Lear and Hamlet being instances of the entity.

o Entities which share common properties are candidates for being converted to generalization hierarchies (See below)

o Entities should not be used to distinguish between time periods. For example, the entities 1st Quarter Profits, 2nd Quarter Profits, etc. should be collapsed into a single entity called Profits. An attribute specifying the time period would be used to categorize by time

o Not every thing the users want to collect information about will be an entity. A complex concept may require more than one entity to represent it. Others "things" users think important may not be entities.

**Attributes**

*Attributes* are data objects that either identify or describe entities. Attributes that identify entities are called *key attributes*. Attributes that describe an entity are called non-key attributes. Key attributes will be discussed in detail in a latter section.

The process for identifying attributes is similar except now you want to look for and extract those names that appear to be descriptive noun phrases.

**Validating Attributes**

Attribute values should be *atomic*, that is, present a single fact. Having disaggregated data allows simpler programming, greater reusability of data, and easier implementation of changes. Normalization also depends upon the "single fact" rule being followed. Common types of violations include:

o Simple aggregation - a common example is Person Name which concatenates first name, middle initial, and last name. Another is Address which concatenates, street address, city, and zip code. When dealing with such attributes, you need to find out if there are good reasons for decomposing them. For example, do the end-

users want to use the person's first name in a form letter? Do they want to sort by zip code?

o Complex codes - these are attributes whose values are codes composed of concatenated pieces of information. An example is the code attached to automobiles and trucks. The code represents over 10 different pieces of information about the vehicle. Unless part of an industry standard, these codes have no meaning to the end user. They are very difficult to process and update.

o Text blocks - these are free-form text fields. While they have a legitimate use, an over reliance on them may indicate that some data requirements are not met by the model.

o Mixed domains - this is where a value of an attribute can have different meaning under different conditions

**Derived Attributes and Code Values**

Two areas where data modeling experts disagree is whether derived attributes and attributes whose values are codes should be permitted in the data model.

Derived attributes are those created by a formula or by a summary operation on other attributes. Arguments against including derived data are based on the premise that derived data should not be stored in a database and therefore should not be included in the data model. The arguments in favor are:

o Derived data is often important to both managers and users and therefore should be included in the data model

o It is just as important, perhaps more so, to document derived attributes just as you would other attributes

o Including derived attributes in the data model does not imply how they will be implemented

A coded value uses one or more letters or numbers to represent a fact. For example, the value Gender might use the letters "M" and "F" as values rather than "Male" and "Female". Those who are against this practice cite that codes have no intuitive meaning to the end-users and add complexity to processing data. Those in favor argue that many organizations have a long history of using coded attributes, that codes save space, and

improve flexibility in that values can be easily added or modified by means of look-up tables.

**Relationships**

Relationships are associations between entities. Typically, a relationship is indicated by a verb connecting two or more entities. For example:

employees are assigned to projects

As relationships are identified they should be classified in terms of cardinality, optionality, direction, and dependence. As a result of defining the relationships, some relationships may be dropped and new relationships added. Cardinality quantifies the relationships between entities by measuring how many instances of one entity are related to a single instance of another. To determine the cardinality, assume the existence of an instance of one of the entities. Then determine how many specific instances of the second entity could be related to the first. Repeat this analysis reversing the entities. For example:

Employees may be assigned to no more than three projects at a time; every project has at least two employees assigned to it.

Here the cardinality of the relationship from employees to projects is three; from projects to employees, the cardinality is two. Therefore, this relationship can be classified as a many-to-many relationship.

If a relationship can have a cardinality of zero, it is an optional relationship. If it must have a cardinality of at least one, the relationship is mandatory. Optional relationships are typically indicated by the conditional tense. For example:

An employee **may** be assigned to a project

Mandatory relationships, on the other hand, are indicated by words such as must have. For example:

A student **must** register for at least three course each semester

In the case of the specific relationship form (1:1 and 1:M), there is always a parent entity and a child entity. In one-to-many relationships, the parent is always the entity with the cardinality of one. In one-to-one relationships, the choice of the parent entity must be made in the context of the business being modeled. If a decision cannot be made, the choice is arbitrary.

**Naming Data Objects**

The names should have the following properties:

o Unique

o Have meaning to the end-user

o Contain the minimum number of words needed to uniquely and accurately describe the object

For entities and attributes, names are singular nouns while relationship names are typically verbs.

Some authors advise against using abbreviations or acronyms because they might lead to confusion about what they mean. Other believe using abbreviations or acronyms are acceptable provided that they are universally used and understood within the organization.

You should also take care to identify and resolve synonyms for entities and attributes. This can happen in large projects where different departments use different terms for the same thing.

**Object Definition**

Complete and accurate definitions are important to make sure that all parties involved in the modeling of the data know exactly what concepts the objects are representing. Definitions should use terms familiar to the user and should precisely explain what the object represents and the role it plays in the enterprise. Some authors recommend having the end-users provide the definitions. If acronyms, or terms not universally understood are used in the definition, then these should be defined .

While defining objects, the modeler should be careful to resolve any instances where a single entity is actually representing two different concepts (homonyms) or where two different entities are actually representing the same "thing" (synonyms). This situation typically arises because individuals or organizations may think about an event or process in terms of their own function.

An example of a homonym would be a case where the Marketing Department defines the entity MARKET in terms of geographical regions while the Sales Departments thinks of this entity in terms of demographics. Unless resolved, the result would be an entity with two different meanings and properties.

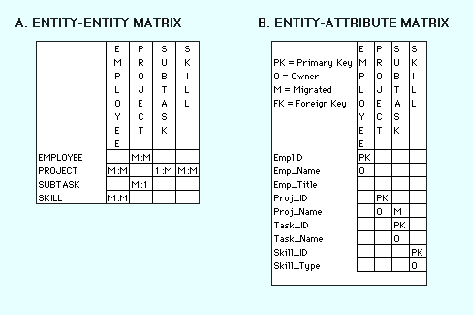
Conversely, an example of a synonym would be the Service Department may have identified an entity called CUSTOMER while the Help Desk has identified the entity CONTACT. In reality, they may mean the same thing, a person who contacts or calls the organization for assistance with a problem. The resolution of synonyms is important in order to avoid redundancy and to avoid possible consistency or integrity problems. **Recording Information in Design Document**

The design document records detailed information about each object used in the model. As you name, define, and describe objects, this information should be placed in this document. If you are not using an automated design tool, the document can be done on paper or with a word processor. There is no standard for the organization of this document but the document should include information about names, definitions, and, for attributes, domains.

Two documents used in the IDEF1X method of modeling are useful for keeping track of objects. These are the ENTITY-ENTITY matrix and the ENTITY-ATTRIBUTE matrix. The ENTITY-ENTITY matrix is a two-dimensional array for indicating relationships between entities. The names of all identified entities are listed along both axes. As relationships are first identified, an "X" is placed in the intersecting points where any of the two axes meet to indicate a possible relationship between the entities involved. As the relationship is further classified, the "X" is replaced with the notation indicating cardinality.

The ENTITY-ATTRIBUTE matrix is used to indicate the assignment of attributes to entities. It is similar in form to the ENTITY-ENTITY matrix except attribute names are listed on the rows.

Figure 2.2 shows examples of an ENTITY-ENTITY matrix and an ENTITY- ATTRIBUTE matrix.

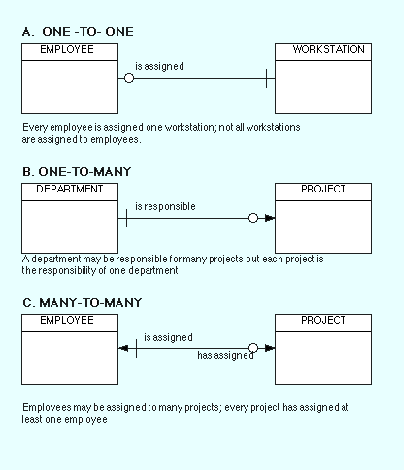


**Developing the Basic Schema**

Once entities and relationships have been identified and defined, the first draft of the entity relationship diagram can be created. This section introduces the ER diagram by demonstrating how to diagram binary relationships. Recursive relationships are also shown.

**Binary Relationships**

Figure 2.3 shows examples of how to diagram one-to-one, one-to-many, and many-to- many relationships.



**One-To-One**

**Figure 2.3 Example of Binary relationships**

Figure 1A shows an example of a one-to-one diagram. Reading the diagram from left to right represents the relationship every employee is assigned a workstation. Because every employee must have a workstation, the symbol for mandatory existence—in this case the crossbar—is placed next to the WORKSTATION entity. Reading from right to left, the diagram shows that not all workstation are assigned to employees. This condition may reflect that some workstations are kept for spares or for loans. Therefore, we use the symbol for optional existence, the circle, next to EMPLOYEE. The cardinality and existence of a relationship must be derived from the "business rules" of the organization. For example, if all workstations owned by an organization were assigned to employees, then the circle would be replaced by a crossbar to indicate mandatory existence. One-to- one relationships are rarely seen in "real-world" data models. Some practioners advise

that most one-to-one relationships should be collapsed into a single entity or converted to a generalization hierarchy.

**One-To-Many**

Figure 1B shows an example of a one-to-many relationship between DEPARTMENT and PROJECT. In this diagram, DEPARTMENT is considered the parent entity while PROJECT is the child. Reading from left to right, the diagram represents departments may be responsible for many projects. The optionality of the relationship reflects the "business rule" that not all departments in the organization will be responsible for managing projects. Reading from right to left, the diagram tells us that every project must be the responsibility of exactly one department.

**Many-To-Many**

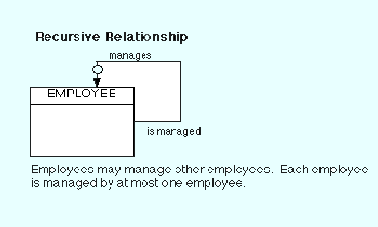
Figure 1C shows a many-to-many relationship between EMPLOYEE and PROJECT. An employee may be assigned to many projects; each project must have many employee Note that the association between EMPLOYEE and PROJECT is optional because, at a given time, an employee may not be assigned to a project. However, the relationship between PROJECT and EMPLOYEE is mandatory because a project must have at least two employees assigned. Many-To-Many relationships can be used in the initial drafting of the model but eventually must be transformed into two one-to-many relationships. The transformation is required because many-to-many relationships cannot be represented by the relational model. The process for resolving many-to-many relationships is discussed in the next section.

**Recursive relationships**

A recursive relationship is an entity is associated with itself. Figure 2.4 shows an example of the recursive relationship.

An employee may manage many employees and each employee is managed by one

employee.



**Summary**

**Example of Recursive relationship**

A data model is a plan for building a database. To be effective, it must be simple enough to communicate to the end user the data structure required by the database yet detailed enough for the database design to use to create the physical structure.

The Entity-Relation Model (ER) is the most common method used to build data models for relational databases.

The Entity-Relationship Model is a conceptual data model that views the real world as consisting of entities and relationships. The model visually represents these concepts by the Entity-Relationship diagram.

The basic constructs of the ER model are entities, relationships, and attributes.

Data modeling must be preceded by planning and analysis.

Planning defines the goals of the database, explains why the goals are important, and sets out the path by which the goals will be reached.

Analysis involves determining the requirements of the database. This is typically done by examining existing documentation and interviewing users.

Data modeling is a bottom up process. A basic model, representing entities and relationships, is developed first. Then detail is added to the model by including information about attributes and business rules.

The first step in creating the data model is to analyze the information gathered during the requirements analysis with the goal of identifying and classifying data objects and relationships

The Entity-Relationship diagram provides a pictorial representation of the major data objects, the entities, and the relationships between them.