Навчальна дисципліна

БАЗИ ДАНИХ

Лектор - к.т.н., доцент

Баклан Ігор Всеволодович

Site: baklaniv.at.ua

E-mail: iaa@ukr.net

2016-2017

Лекція №3.

Реляційна модель даних

- Basic Concepts
- Domains
- Relations
- Relational Database System
- Identifying, Representing and Implementing Relationships
- Relation-Attributes List and Relationship List
- Non-relational Approaches
- Summary and Concluding Remarks

3.1 Basic Concepts

The relational model is by far the most widely used model for database design. The model owes its success to the fact that it is firmly founded on mathematical principles (set theory and linear algebra) which have been tested and proven; like the underlying principles, the model itself has been tested and proven oven the years. Before we can proceed, there are some fundamental concepts to be introduced (Figure 3-1):

Entity: An object, concept or thing about which data is stored. Examples include **PurchaseOrder**, **Person**, **Course**, **Department**, **Program**, **Student**. Entities are implemented as two-dimensional tables and ultimately files.

Attributes: Some qualities associated with the entity; e.g. **Order#**, **OrderDate** and **Item#** of entity **PurchaseOrder**; **Dept#** & **DeptName** of entity **Department**. Two synonymous terms for attributes are *elements* and *properties* ; they correspond to columns of a table and are ultimately implemented as fields of a record.

Entity Set: A set of related entities.

Relationship: An inherent mapping involving two or more entities. Relationships are represented in *relations*.

Relation: A two-dimensional (tabular) representation of entities and relationships. A binary relation contains two attributes; an *n*-ary relation contains n attributes. A binary relationship involves an association between two entities; an n-ary relationship involves an association among n entities. For a more formal definition, see section 3.3.

Tuples: Correspond to rows of the table, or records of a file.

Primary Key: An attribute or combination of attributes for which values uniquely identify tuples in the relation. The primary key is chosen from a set of *candidate keys*.

Candidate Keys: There may be more than one potential keys of a relation. Each is called a candidate key.

Alternate Key: A candidate key that is not the primary key.

Foreign Key: An attribute (or combination of attributes) that is primary key in another relation.

Domain: A pool of all legal values from which actual attribute values are drawn.

Cardinality: Number of tuples in a relation. The cardinality varies with time.

Degree: Number of attributes in a relation; also called the *arity*. Degree is also used to describe the number of entities implied by a relationship.

Figure 3-1. *Basic Concepts*

Figure 3-2 provides a list of commonly used relational terms and their informal equivalents:

Formal Relational Term	Informal Equivalents
Entity	Object conceptualized as a table, implemented as a file
Relation	As for entity
Tuple	Conceptualized as a row, implemented as a record
Attribute	Conceptualized as a column, implemented as a field
Cardinality	Conceptualized as the number of rows
Degree	Conceptualized as the number of columns
Domain	Conceptualized as a pool of legal values

Figure 3-2. *Relational Terms and their Informal Equivalents*

Figure 3-3 illustrates how the terms are applied. Observe that the idea of *entity* and *relation* seem to be similar. This will be clarified later. For now, assume that similarity.

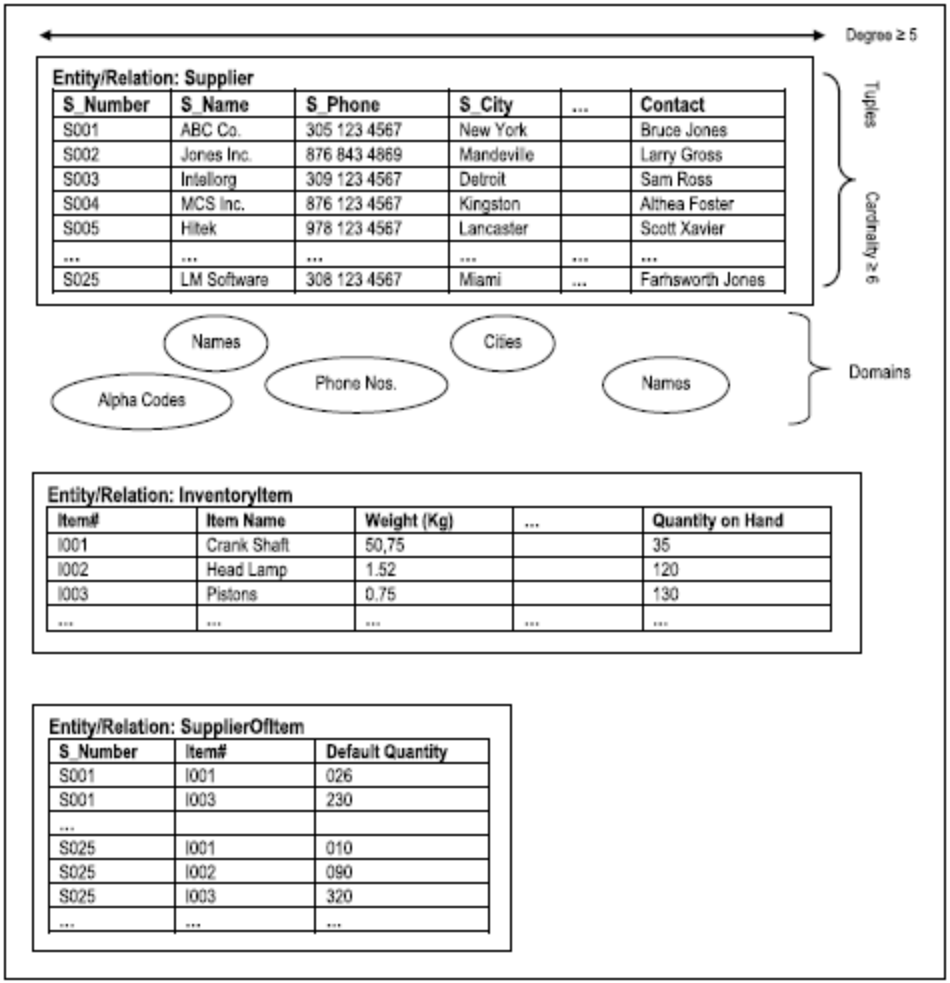

Figure 3-3. Illustrating Basic Terms

3.2 Domains

A domain is a named set of *scalar values* from which attribute values are drawn. Scalar values are non-decomposable (atomic) values.

Each attribute of a relation must be defined on an underlying domain. The attribute values must come from the underlying domain (as illustrated in Figure 3-3).

Domains are conceptual; they may not be (and usually are not) actually stored in the database. Rather, the subsets of the domains containing actual attribute values are stored. Domains are analogous to data types in high level programming languages such as Pascal, C++, Java, etc.

A *composite domain* is a combination of simple domains. Whether a composite domain is used or is replaced by its constituent simple domains, is a design decision that should follow thoughtful consideration.

Example 1: Date is an excellent illustration of a composite domain, as explained below:

Date is a combination of
Year which has range 0 .. 9999
Month which has range 1 .. 12
Day which has range 1 .. 31

This domain therefore has a total of 12 * 31 * 10,000 values, but not all values are valid dates.

There can therefore be composite attributes. Composite domains are analogous to Pascal records and C++ structures. Few systems support composite domains and composite attributes.

Significance of Domains

An understanding of domains is critical for the following reason: If attributes of different relations (entities) come from the same domain, then comparisons can be made; otherwise, comparisons are meaningless.

Example 2: The following illustrations should emphasize the importance of domains:

```
// Referring to figure 3.3 and using SQL statements on relations InventoryItem and SupplierOfItem:

// The following SQL statement is valid:
SELECT * FROM SupplierOfItems SI, Supplier S WHERE SI.S_Number = S.S_Number;

/* The following SQL statement is not a valid SQL statement since attempt is being made to compare
   attributes (Weight and DefaultQuantity) defined on different domains:*/
SELECT * FROM SupplierOfItems SI, InventoryItem I WHERE SI.DefaultQuantitty = I.Weight;
```

3.3 Relations

A relation R on a collection of domains D1, D2, . . . Dn *(*not necessarily distinct*)* consists of two parts — a *heading* and a *body*.

The heading consists of a fixed set of attributes, or more precisely, attribute-domain pairs,

$$\{(A1:D1), (A2:D2), \ldots (An:Dn)\}$$

such that each attribute corresponds to exactly one domain and n is the degree of the relation. Another term used to describe the heading of a relation is the *predicate* of the relation.

The body consists of a time-varying set of tuples where each tuple consists of a set of attribute-value pairs

$$\{(A1:Vi1), (A2:Vi2), \ldots (An:Vin)\}\ (i = 1 \ldots m)$$

where m is the number of tuples *(cardinality)* in the set. The body of the relation is also sometimes referred to as the *proposition* of the relation. The proposition defines a set of tuples whereby for each row in the relation, the respective attributes take on legal values from their respective domains.

Observe that the definition of a relation appears to be similar to that of an entity. There are two subtle differences:

- The term *relation* as used, belongs to the field of relational systems. We talk about relations because we are discussing the relational model. Entities on the other hand, describe identifiable objects and/or situations.
- *Entity*, as defined does not account for relationships. Relation on the other hand, accounts for entities as well as relationships. Thus in the relational model, we represent entities as relations and (M:M) relationships (between entities) as relations. A binary relation for instance, has two attributes. If both attributes are foreign keys and they both constitute the primary key, this binary relation actually represents a *many-to-many relationship* between two referenced relations; otherwise it is (a relation that can be construed as) an entity. This point will become clear as we proceed.

The foregoing underscores the point that entities can be construed as special kinds of relations. In designing a database, the software engineer or database designer commences by identifying entities during the requirements specification. After further analysis, these entities are eventually implemented by normalized relations.

Note: A unary relation differs from a domain in the sense that former is dynamic and the latter static.

3.3.1 Properties of a Relation

Based on the relational model, all relations have the following properties:

- No duplicate tuples (records)
- Records are unordered
- Attributes are unordered
- Attribute values are atomic

The first and last properties are constraints that both end users and software engineers should be cognizant of, since they have to manage data contained in the database; they are also of interest to the database designer. The second and third properties on the surface are immaterial to end users as well as software engineers; they are usually enforced by the DBMS in a manner that is transparent to the end user. However, when the DBMS is written, concern has to be given to accessing of records. Further, DBMS suites are typically written to give the illusion that the attributes of a relation are ordered.

3.3.2 Kinds of Relations

A database will consist of various types of relations, some of them at different stages of the system. The common categories of relations are mentioned below:

1. **Base Relations** are named and permanently represented in the database. They make up the conceptual schema of the database; they form the foundation of the database.

2. **Views** (virtual relations) are derived from named (base) relations. A view stores the definition of the virtual relation (derived from base relations), but stores no physical data. It is simply a logical (conceptual/external) interpretation of data stored in base relations. SQL views and System i logical files are good examples of views.

3. **Snapshots** are named, derived relations. They differ from logical views in that they are represented in terms of definition as well as physically stored data. From the perspective of the end user, a snapshot relation is typically (but not necessarily) read-only. To illustrate, consider two systems — System-A and System-B — which both need to access a database table, Table-X. Suppose that System-A has update rights to Table-X, but System-B does not. Table-X is therefore stored in System-A's database; a duplicate version for read-only purposes, is stored in System-B, and is periodically updated (without user interference) from System-A.

4. **Query Results:** Queries are typically entered at a command prompt (they may be also embedded in high level language programs or stored in special query files). Results may be directed to screen, printer, or a named relation. An important principle to note is that a query when executed always results in a new relation. This principle will be elucidated later in the course.

5. **Intermediate Results:** The DBMS may create an intermediate relation to assist in furnishing a final answer to a complex query request. This will also be elucidated later in the course.

6. **Temporary Relations** are named relations that are destroyed at some point in time.

3.4 Relational Database System

A *relational database system* (RDBS) is a collection of time-varying *normalized relations*, managed through an appropriate user interface, and with desirable constraints and features that enhance the effective, efficient management of the database. These desirable features and constraints will be discussed (see chapter 9) as we progress through the course. The term *normalized relations* will be fully clarified in chapter 4; for now, just consider it to mean that the relations are designed to promote efficiency and accessibility.

The relations are conceptualized as tables and ultimately implemented as files. Each relation contains one and only one record type. Each relation has a primary key (chosen from a set of candidate keys). In many cases, the primary key is obvious and can be identified intuitively. In situations where this is not the case, the database designer, based on principles to be discussed in the next chapter, typically takes decision about the primary key.

Each record type is made up of *atomic attributes*. This means that each attribute is defined on a single domain, and can only have a value from that domain. Moreover, when data is loaded into the database, each record from any given table has a unique primary key value.

Superimposed on the database is a user interface that facilitates access of the database by end users. The database and the user interface are designed to ensure that certain objectives are met (section 1.2) and established standards are conformed to.

Steps in Building a Relational Database System

In constructing a RDBS, the following steps may be pursued:

a. Identify entities

b. Identify relationships

c. Eliminate unnecessary relationships

d. Develop *entity-relationship diagram* (ERD), *object-relationship diagram* (ORD) or some equivalent model

e. Normalize the database

f. Revise E-R diagram, O-R diagram, or the equivalent model used

g. Design the user interface

h. Proceed to development phase

3.5 Identifying, Representing, and Implementing Relationships

As mentioned earlier, a relationship is an inherent mapping involving two or more relations. In planning a relational database, it is very important to know how to identify and represent relationships. Of course, the ultimate objective is successful implementation of the model. Let us take some time to discuss these issues:

3.5.1 Identifying Relationships

To identify relationships, you have to know what a relationship is (review section 3.1) and what types of relationships there are. There are six types of relationships:

- One-to-one (1:1) Relationship
- One-to-many (1:M) Relationship
- Many-to-one (M:1) Relationship
- Many-to-many (M:M) Relationship
- Component Relationship
- Subtype Relationship

The first four types of relationships are referred to as *traditional* relationships because up until object model (for database design) gained preeminence, they were essentially the kinds of relationships that were facilitated by the relational model. Observe also, that the only difference between a 1:M relationship and an M:1 relation is a matter of perspective; thus, a 1:M relationship may also be described as an M:1 relationship (so that in practice, there are really three types of traditional relationships). Put another way:

> If R1, R2 are two relations and there is a 1:M relationship between R1 and R2, an alternate way of describing this situation is to say that there is an M:1 relationship between R2 and R1.

For traditional relationships, to determine the type of relationship between two relations (entities) R1 and R2, ask and determine the answer to the following questions:

- How many records of R1 can reference a single record of R2?
- How many records of R2 can reference a single record of R1?

To test for a component relationship between any two relations R1 and R2, ask and determine the answer to the following questions:

- Is (a record of) R1 composed of (a record of) R2?
- Is (a record of) R2 composed of (a record of) R1?

For a subtype relationship, the test is a bit more detailed; for relations R1 and R2, ask and determine the answer to the following questions:

- Is (a record of) R1 also a (a record of) R2?
- Is (a record of) R2 also a (a record of) R1?

Possible answers to these questions are always, sometimes, or never. The possibilities are shown below:

R1 always R2, R2 always R1	⇒	R1 and R2 are synonymous
R1 always R2, R2 sometimes R1	⇒	R1 is a subtype of R2
R1 always R2, R2 never R1	⇒	Makes no sense
R1 sometimes R2, R2 always R1	⇒	R2 is a sub-type of R1
R1 sometimes R2, R2 sometimes R1	⇒	Inconclusive
R1 sometimes R2, R2 never R1	⇒	Makes no sense
R1 never R2, R2 always R1	⇒	Makes no sense
R1 never R2, R2 sometimes R1	⇒	Makes no sense
R1 never R2, R2 never R1	⇒	No subtype relationship

3.5.2 Representing Relationships

Having identified the entities and relationships, the next logical question is, how do we represent them? Four approaches have been used: *database hierarchies, simple networks, complex networks,* the *entity-relationship model* and the *object-relationship model*. The first three approaches are traditional approaches that have made way for the more reputed latter two approaches. We will therefore start by discussing the latter two approaches.

The Entity-Relationship Model

The popular answer to this challenge of database representation is the *entity-relationship diagram* (ERD or E-R diagram). Figure 3-4a shows the symbols used in an ERD, while Figure 3-4b provides an illustration based on the Crows-Foot notation. In the diagram, the convention to show attributes of each entity has been relaxed, thus avoiding clutter. Note also that relationships are labeled as verbs so that in linking one entity to another, one can read an entity-verb-entity formulation. If the verb is on the right or above the relationship line, the convention is to read from top-to-bottom or left-to-right. If the verb is on the left or below the relationship line, the convention is to read from bottom-to-top or right-to-left.

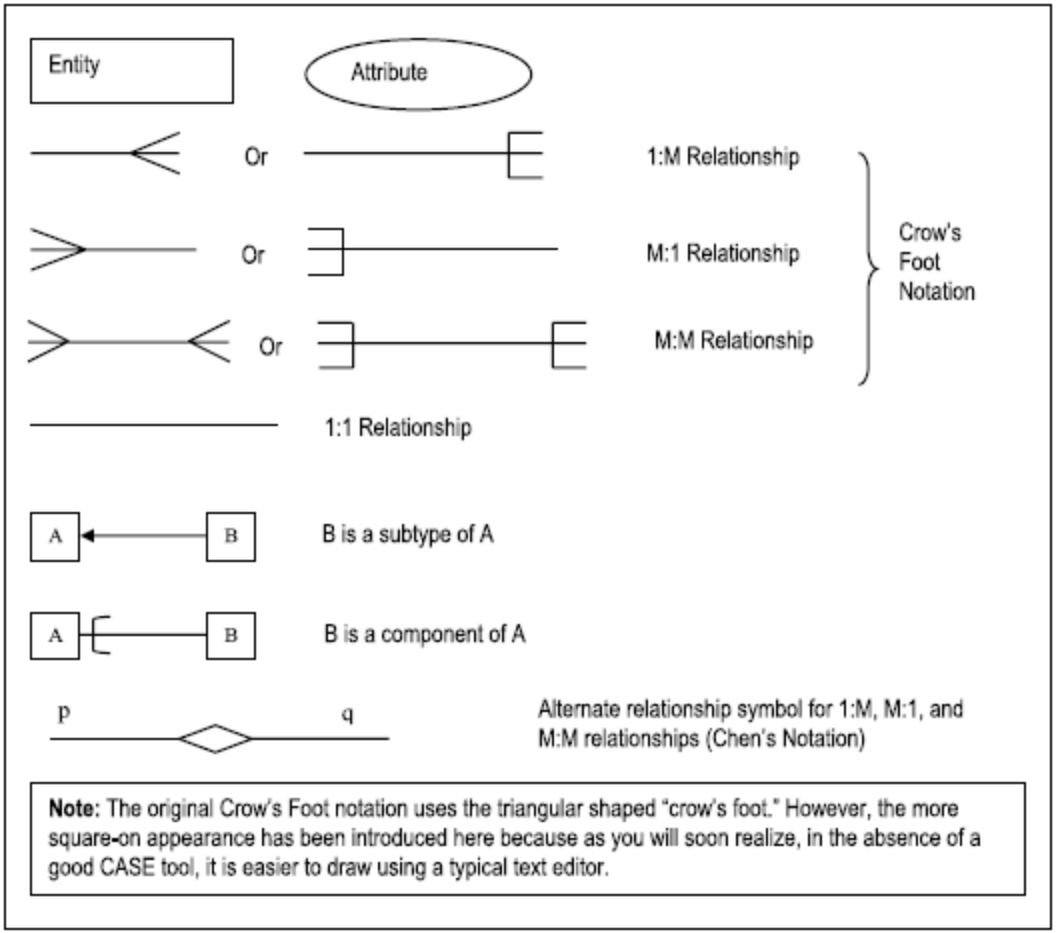

Figure 3-4a. *Symbols Used in E-R Diagrams*

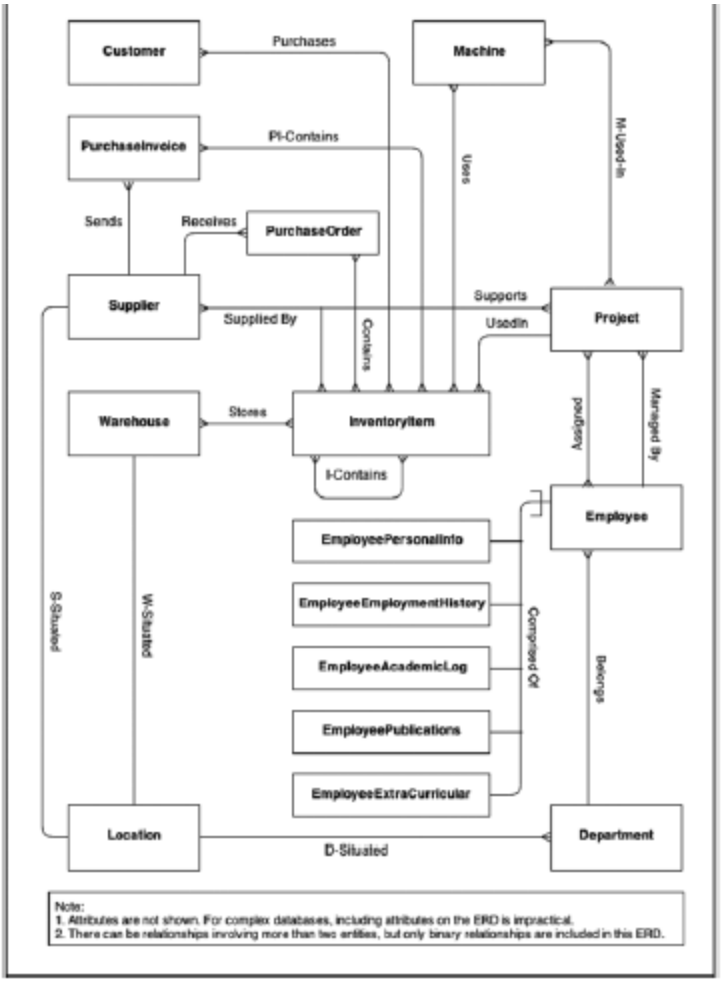

Figure 3-4b. *Partial ERD for Manufacturing Firm*

The ERD is normally used to show binary relationships but can also show n-ary relationships. In many cases, E-R diagrams show only binary relationships. For example, a possible ternary relationship not shown in Figure 3-4b is **Supplier-Schedule** (linking **Supplier**, **InventoryItem** and **Project**). The reason for this is the following principle:

> All relationships of degree greater than 2 can be decomposed to a set of binary relationships. This may or may not be required.

The proof for this principle is beyond the scope of this course. However, we shall revisit it later, and provide additional clarifications. For now, a simple illustration will suffice: Figure 3-5 shows how the ternary **Supplier-Schedule** relationship may be broken down into three binary relationships. Since care must be taken in applying this principle, it will be further discussed in the next chapter.

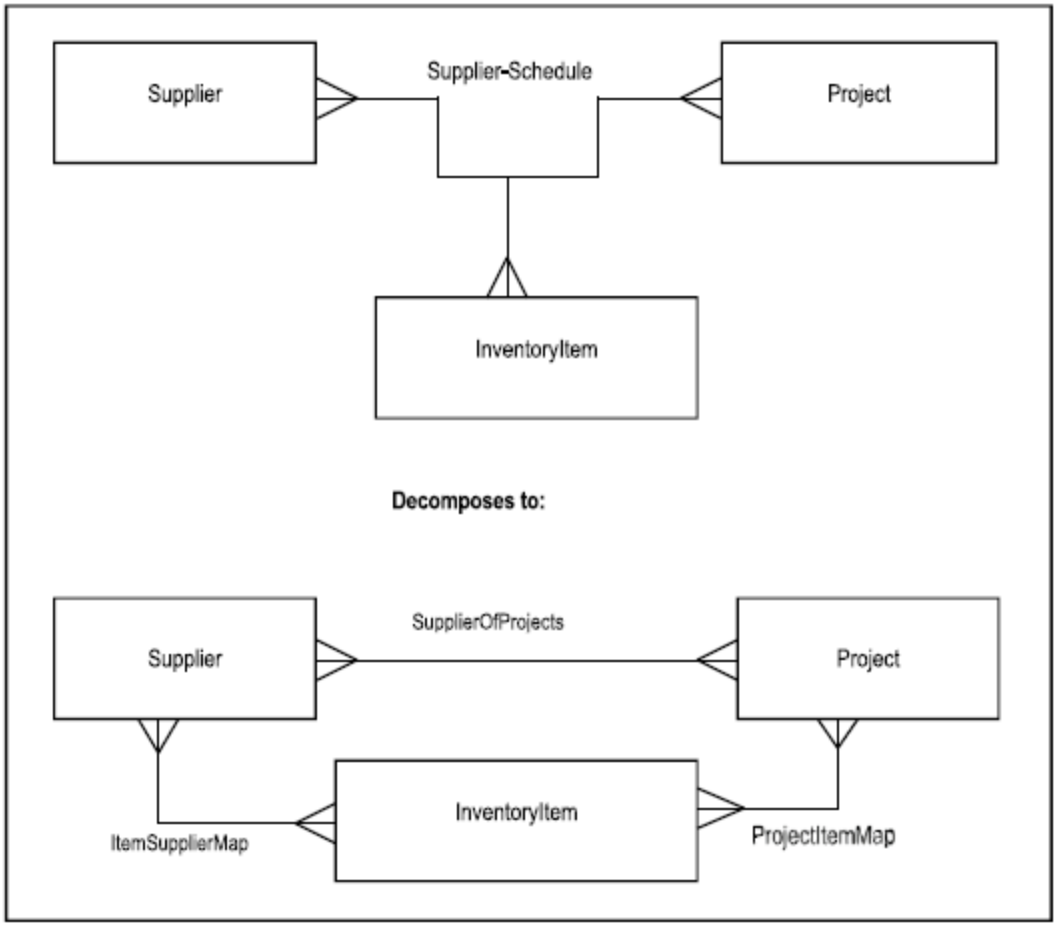

Figure 3-5. *Decomposing a Ternary Relationship*

The Object-Relationship Model

As you are aware, or will soon learn (from your software engineering course), there are, broadly speaking, two alternate paradigms for software construction: the functional approach (which is the traditional approach) and the object-oriented (OO) approach. In an object-oriented environment, the comparative methodology for the E-R diagram is the *object-relationship diagram* (ORD or O-R diagram). The concept of an ORD is similar to that of an ERD, and the diagrams are also similar, but there are a few exceptions:

- In the OO paradigm, the *object type* replaces the entity (type) of the relational model. Like the entity, an object type is a concept or thing about which data is stored. Additionally, the object type defines a set of operations, which will be applicable to all objects (instances) of that type.
- The symbol used to denote an object type is similar to an entity symbol, except that it has two extended areas — one for the attributes of the object type, and the other for its defined operations.
- The preferred diagramming convention is the UML (Unified Modeling Language) notation.
- Depending on the OO development tool, there might be additional notations regarding the cardinality (more precisely, multiplicity) of the relationships represented.

A full treatment of the OO approach is beyond the scope of this course. You are no doubt familiar with using UML diagrams in your OO programming courses. For a quick review of the fundamentals, please see references [Lee, 2002] and [Martin, 1993]. However, in the interest of comprehensive coverage, an overview of the approach is provided in chapters 5 and 23.

Database Tree

A *database tree* (hierarchy) is a traditional alternative, which used to be employed prior to the introduction or the E-R model; it was successfully employed in a system called RAMIS (the original acronym stands for "Random Access Management Information System"). A database tree (hierarchy) is a collection of entities and 1:M relationships arranged such that the following conditions hold:

- The root has no parent
- Each subsequent node has a single parent

Figure 3-6 illustrates a database hierarchy. Observe that it looks like a general tree (review your data structures). Except for the root (node A), each node has a parent node that it references. Note also that all the relationships are 1:M relationships (traditionally referred to as *parent-child relationships*).

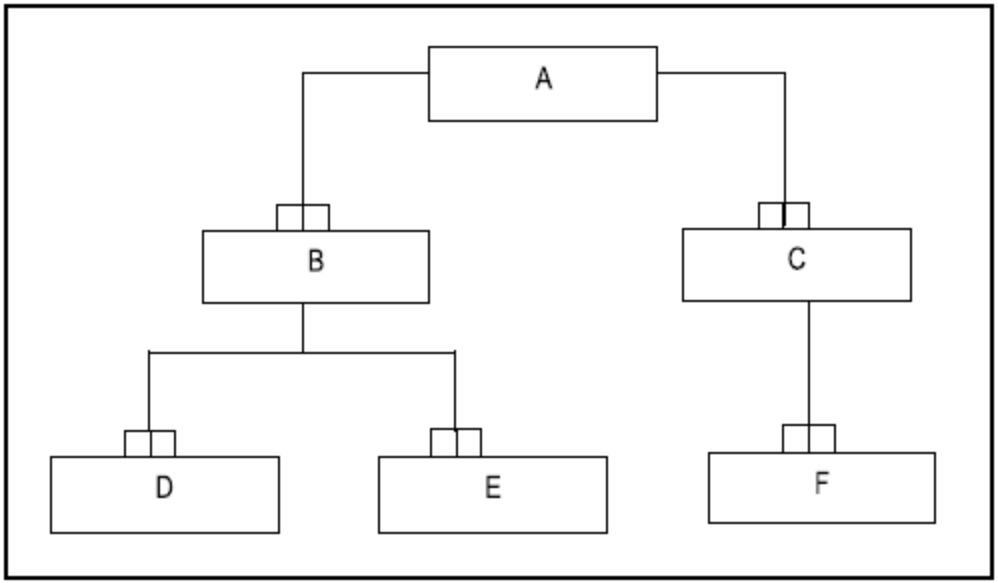

Figure 3-6. *Example of a Hierarchy (Tree)*

Database Networks

The database network approach is another traditional approach that is no longer employed. In the interest of historical context, a brief overview is provided here. A *simple database network* is a collection of entities and 1:M relationships arranged such that any member can have multiple parents, providing that the parents are different entities. Figure 3-7 illustrates the approach. It was successfully employed in a DBMS called the CODASYL system (the original CODASYL acronym stands for "Conference on Data Systems Languages").

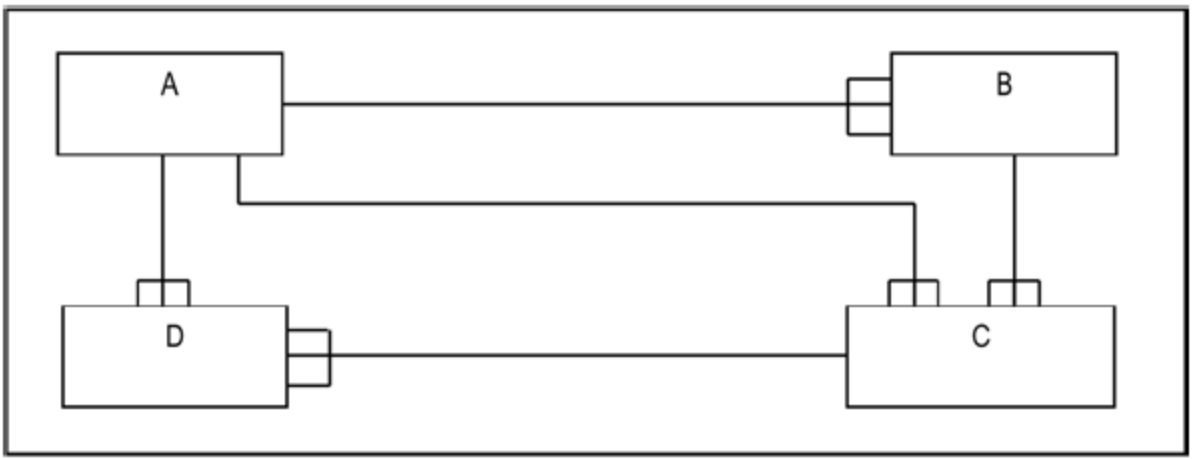

Figure 3-7. *A Simple Network*

A *complex database network* is a collection of entities and relationships, at least one of the relationships being an M:M relationship. Figure 3-8 illustrates. The complex network can be reduced to a simple network by replacing all M:M relationships with M:1 relationships. The technique for replacing M:M relationships will be discussed in the upcoming subsection.

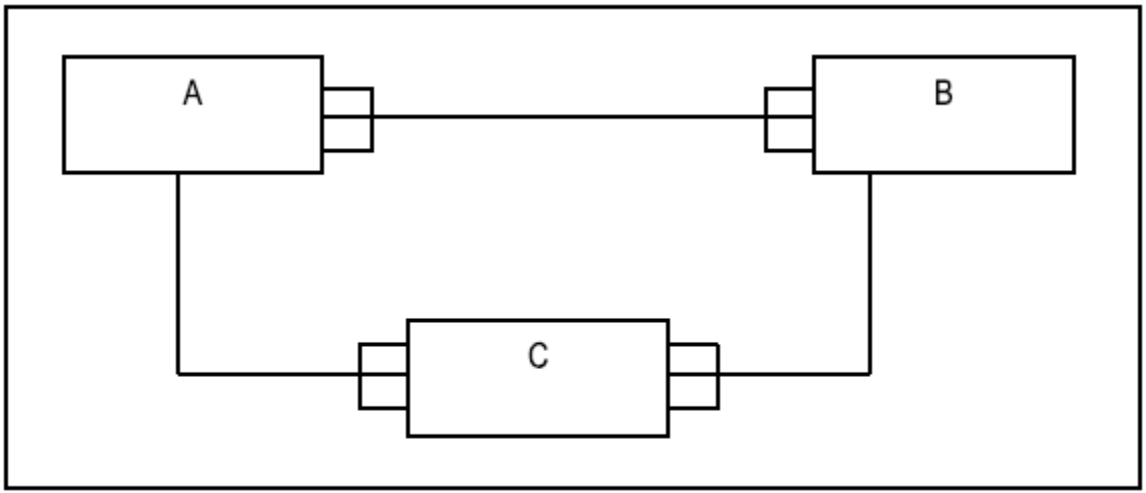

Figure 3-8. *Complex Network*

3.5.3 Multiplicity of Relationships

It is customary to indicate on the ERD (or ORD), the *multiplicity* (also called the *cardinality*) of each relationship. By this we mean, how many occurrences of one entity (or object type) can be associated with one occurrence of the other entity (or object type). This information is particularly useful when the system is being constructed. Moreover, violation of multiplicity constraints could put the integrity of the system is question, which of course is undesirable. Usually, the DBMS does not facilitate enforcement of multiplicity constraints at the database level. Rather, they are typically enforced at the application level by the software engineer.

Several notations for multiplicity have been proposed, but the Chen notation (first published in 1976, and reiterated in [Chen, 1994]) is particularly clear; it is paraphrased here: Place beside each entity (or object type), two numbers [*x*,*y*]. The first number (*x*) indicates the minimum participation, while the second (*y*) indicates the maximum participation.

An alternate notation is to use two additional symbols along with the Crow's Foot notation: an open circle to indicate a participation of zero, and a stroke (|) to indicate a participation of 1. The maximum participation is always indicated nearest to the entity (or object type) box.

For convenience, you could also use the Chen's notation for multiplicity, along with the Crow's Foot notation for representing the relationships. The Chen notation is preferred because of its clarity and the amount of information it conveys. Figure 3-9 provides an illustrative comparison of the two notations.

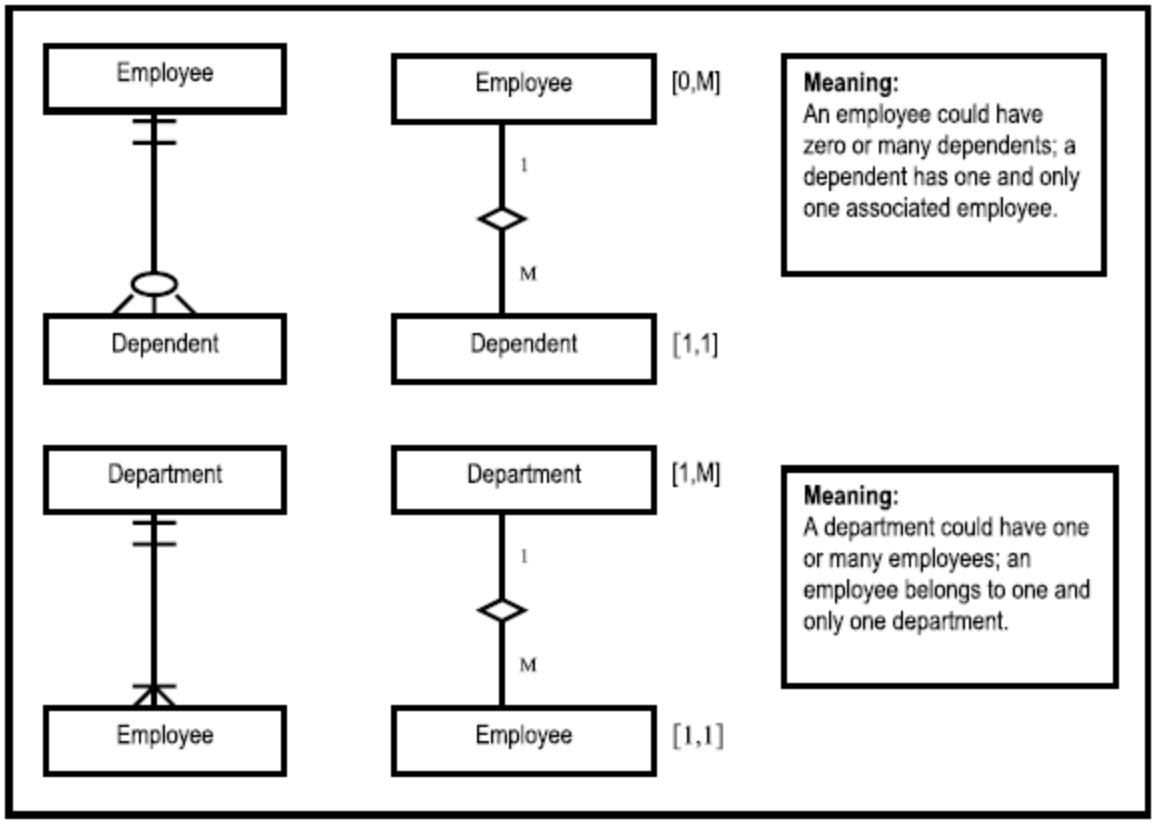

Figure 3-9. *Illustrating Multiplicity Notations*

3.5.4 Implementing Relationships

Assuming the E-R model, relationships can be implemented by following a set of guidelines as outlined below:

To implement a 1:M relationship, store the primary key of one as a foreign key of the other (foreign key must be on the "many side"). Figures 3.13 and 3.14 illustrate (there is a M:1 relationship between **Suppliers** and **Locations**; there are others that you should identify).

To implement an M:M relationship, introduce a third intersecting (1:M) relation. The new relation is usually keyed on all the foreign keys (or a *surrogate #*). Also, the original relations/entities form 1:M relationships with the intersecting relation (figure 3.10 illustrates).

To implement a subtype relationship, introduce a foreign key in the subtype, which is the primary key in the referenced super-type. Further, make the foreign key in the subtype, the primary key of that subtype. In the case of *multiple inheritance* (where a subtype has more than one super-types), make the introduced foreign keys in the subtype, candidate keys, one of which will be the primary key. Figures 3.11 and 3.12 illustrate this strategy.

To implement a component relationship, introduce in the component relation, a foreign key that is the primary key in the summary relation. This foreign key will form part of the primary key (or a candidate key) in the component relation. Figures 3.11 and 3.12 illustrate this strategy.

To implement a 1:1 relationship, introduce a foreign key in one relation (preferably the primary relation) such that the primary key of one is an attribute in the other. Then enforce a constraint that forbids multiple foreign keys referencing a single primary key. Alternately, treat the 1:1 relationship as a subtype relationship (but ignore enforcing inheritance).

The foregoing strategies should underscore forcefully in your mind, the importance of foreign keys in database design. In fact, foreign keys are referred to as the "glue" that holds the database together. We shall revisit this concept in the next chapter.

In many textbooks and database environment, you will see and/or hear the term *parent-child relationship*. This is a rather lame term, borrowed from preexisting hierarchical database systems, to describe 1:1 and 1:M relationships. In a parent-child relationship, the *parent relation* is the referenced relation; the child relation is the *referencing relation*. Throughout this course, these terms are avoided because they are rather confusing, and do not accurately describe several scenarios involving 1:1 and/or 1:M relationships. Alternately, we will use no euphemism for 1:1 and 1:M relationships; instead of parent relation, we'll say the *referenced relation*; instead of child relation, we say *primary relation* or *referencing relation*.

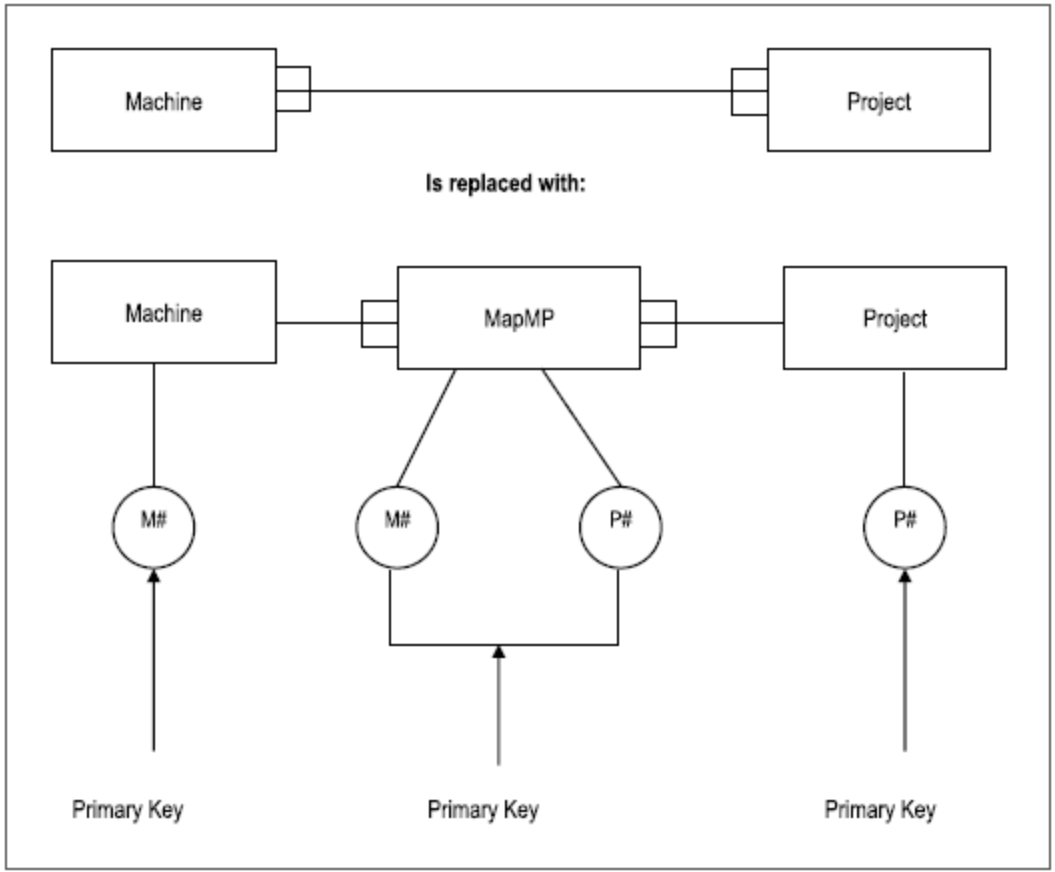

Figure 3-10. *Implementing M:M Relationships*

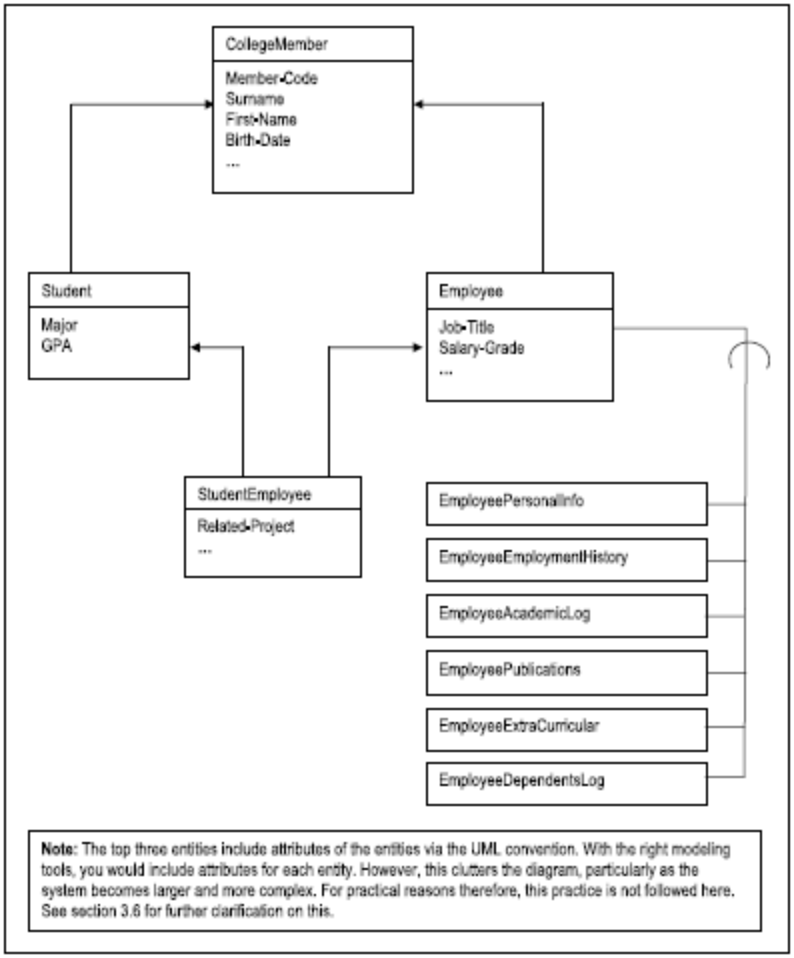

Figure 3-11. *Illustrating Subtype and Component Relationships*

Relation	Attributes
College Member	*MemberCode*, Surname, First-Name, BirthDate, …
Student	*MemberCode*, Major, GPA, …
Employee	*MemberCode*, JobTitle, SalaryGrade, …
StudentEmployee	*MemberCode*, Related-Project
EmployeePersonalInfo	*MemberCode*, Address, Telephone, …
EmployeeEmploymentHistory	*MemberCode, JobSequence*, Organization, …
EmployeeAcademicLog	*MemberCode, LogSequence*, Institution, Period-Attended, Award, …
EmployeePublications	*MemberCode, PublCode*, Title, Book-Journal-Flag, …
EmployeeExtraCurricular	*MemberCode, ActivityCode*, Activity-Description, …

Note:

1. Primary key attributes and foreign key attributes are in italics.
2. This is not a comprehensive RAL. For several of the relations included, there are additional attributes to be added (indicated by the three periods in the attributes column).

Figure 3-12. *Illustrating the Implementation of Subtype and Component Relationships*

3.6 The Relation-Attributes List and Relationship List

In large, complex information systems projects, it is often impractical to attempt to develop and maintain ERDs, unless they are automatically generated and maintained by computer-aided software engineering (CASE) tools (more on these in chapter 5). Even when maintained by CASE tools, an ERD for such a project could become large, spanning several pages. Reading and interpretation then becomes difficult.

To circumvent the above challenges, a *Relation-Attribute List* (RAL) and a *Relationship List* (RL) may be constructed and maintained. The former maintains information on all relations of the system and the latter maintains information on all relationships implemented in the system. Figures 3-13 and 3-14 illustrate partial RAL and RL for the database model of Figure 3-4b. As you examine these figures, please note the following:

- In practice, the format of the RL shown in Figure 3-14b is used as the final list over the format shown in Figure 3-14a (the format used in Figure 3-14a can be deduced by simply identifying all possible relationships among entities; hence, it may contain optional relationships and is therefore useful as a first draft).
- The revised RL of Figure 3-14b has been stripped of all M:M relationships (review section 3.5.4 on treating M:M relationships).
- The relations **PurchaseInvSummary** and **PurchaseInvDetail** of Figure 3-13, are used to replace the M:M relationship between **PurchaseInvoice** and **InventoryItem** in Figure 3-4b. Similarly, the relations **PurchaseOrdSummary** and **PurchaseOrdDetail** are used to replace the M:M relationship between **PurchaseOrder** and **InventoryItem** in Figure 3-4b.

- In constructing the RAL and RL, it is sometimes useful to use the RL to refine the RAL and vice versa. In particular, once you have identified all the (mandatory) relationships, you may use this along with the principles outlined in section 3.5.4 to refine the RAL.
- Remember, the model of Figure 3-4b, RAL of Figure 3-13, and the RL of Figure 3-14 do not represent a comprehensive coverage of the database requirements of a manufacturing firm; neither are they intended to be. Rather, they serve as useful illustrations.

Relation	Attributes
Customer	*Cust#*, CustName, Address,..., Reference Person, ...
Supplier	*Supp#*, SuppName, Address, SuppLoc#, ...
Machine	*Mach#*, Mach-Description, ...
Project	*Proj#*, ProjName, *ProjManagerEmp#*, ...
Warehouse	*Whouse#*, WhouseName, WhouseSize, *WhouseLoc#*, ...
InventoryItem	*Item#*, ItemName, ...
Location	*Loc#*, LocationName, DistanceFromHQ, ...
Department	*Dept#*, DeptName, *DeptLoc#*, ...
Employee	*Emp#*, EmpName, *EmpProj#*, *EmpDept#*, DOB,...
PurchaseOrdSummary	*OrderRef*, Order#, OrderDate, *OrderSupp#*, OrderStatus, ...
PurchaseOrdDetail	*POOrderRef*, *OrderItem#*, OrderQuantity, OrderUnitPrice
PurchaseInvSummary	*PurchaseRef*, Invoice#, *InvSupp#*, *InvOrderRef*, InvDate, InvAmount, InvStatus, ...
PurchaseInvDetail	*PIDPurchaseRef*, *PIDItem#*, PIDItemQuantity, PIDItemUnitPrice
SaleInvSummary	*SaleRef*, SIInvoice#, SaleDate, *SaleCust#*, InvoiceStatus, SaleAmount, ...
SaleInvDetail	*SIDSaleRef*, *SaleItem#*, Quantity, UnitPrice
MachineUsage	*MUMach#*, *MUItem#*
MachProjects	*MPMach#*, *MPProj#*
ProjSupp	*PSSupp#*, *PSProj#*
ItemProj	*IPItem#*, *IPProj#*
SuppItems	*SISupp#*, *SIItem#*
Stock	*SWhouse#*, *SItem#*
ItemStruct	*ISThisItem#*, *ISComptItem#*
EmployeePersonalInfo	*EPIEmp#*, MaritalStatus, Address, Telephone, Email, ...
EmployeeEmploymentHistory	*EHEmp#*, *EHJobSequence*, Organization, Title, ...
EmployeeAcademicLog	*ALEmp#*, *ALLogSequence*, Institution, PeriodAttended, Award, ...
EmployeePublications	*EPEmp#*, *PubiCode*, Title, BookJournalFlag, ...
EmployeeExtraCurricular	*EXEmp#*, *EXActivityCode*, ActivityDescription, ...
...	

Note:

1. *Primary key attributes and foreign key attributes are in italics.*
2. This is not a comprehensive RAL. For several of the relations included, there are additional attributes to be added (indicated by the three periods in the attributes column). Also, additional relations would be required in order to have a comprehensive model (indicated by the three periods in the previous row).
3. For each relation, an effort is made to keep attribute names unique (to the entire database), even if the attribute is a foreign key. For instance, in the relation **Employee**, the attribute **EmpDept#** is a foreign key that references **Dept#** in the relation **Department**. This convention applies for all foreign keys.
4. The attributes **OrderRef** (in **PurchaseOrdSummary** relation), **PurchaseRef** (in **PurchaseInvSummary** relation) and **SaleRef** (in **SaleInvSummary** relation) are examples of surrogates. Surrogates will be more thoroughly discussed in chapter 5.

Figure 3-13. *Partial Relation-Attributes List for a Manufacturing Firm's Database*

Relationship Name	Participating Relations	Type	Comment
SuppliedBy	Supplier, InventoryItem	M:M	Mandatory
Purchases	Customer, InventoryItem	M:M	Mandatory
Uses	Machine, InventoryItem	M:M	Mandatory
Used-In	InventoryItem, Project	M:M	Mandatory
M-Used-In	Machine, Project	M:M	Mandatory
Stores	Warehouse, InventoryItem	M:M	Mandatory
Contains	InventoryItem, InventoryItem	M:M	Mandatory
Assigned	Employee, Project	M:1	Mandatory
Belongs	Employee, Department	M:1	Mandatory
D-Situated	Location, Department	1:M	Mandatory
W-Situated	Warehouse, Location	1:1	Mandatory
S-Situated	Supplier, Location	1:1	Mandatory
Sends	Supplier, PurchaseInvoice	M:M	Mandatory
I-Contains	PurchaseInvoice, InventoryItem	M:M	Mandatory
RequestedOn	PurchaseOrder, InventoryItem	M:M	Mandatory
Receives	Supplier, PurchaseOrder	1:M	Mandatory
Supports	Supplier, Project	M:M	Mandatory
SupplierSchedule	Supplier, InventoryItem, Project	M:M	Optional
ComposedOf	Employee, EmployeePersonalInfo, EmployeeEmploymentHistory, EmployeeAcademicLog, EmployeePublications, EmployeeExtraCurricular	Comp	Mandatory

Figure 3-14a. *Relationships List for a Manufacturing Firm's Database*

Named Relation	Referenced Relations	Type	Comment
SaleInvSummary	Customer	M:1	Implements relationship Purchases
SaleInvDetail	SaleInvSummary	M:1	Implements relationship Purchases
	InventoryItem	M:1	Implements relationship Purchases
PurchaseInvSummary	Supplier	M:1	Implements relationships Sends and ▶Contains
PurchaseInvDetail	PurchaseInvSummary	M:1	Implements relationship ▶Contains
	InventoryItem	M:1	Implements relationship ▶Contains
MachineUsage	Machine	M:1	Implements relationship Uses
	InventoryItem	M:1	Implements relationship Uses
MachProjects	Machine	M:1	Implements relationship M-Used-In
	Project	M:1	Implements relationship M◀Used◀In
PurchaseOrdSummary	Supplier	M:1	Implements relationships Receives and RequestedOn
PurchaseOrdDetail	PurchaseOrdSummary	M:1	Implements relationship RequestedOn
	InventoryItem	M:1	Implements relationship RequestedOn
Supplier	Location	1:1	Implements relationship S◀Situated
SuppItems	Supplier	M:1	Implements relationship SuppliedBy
	InventoryItem	M:1	Implements relationship SuppliedBy
ProjSupp	Supplier	M:1	Implements relationship Supports
	Project	M:1	Implements relationship Supports
ItemProj	Project	M:1	Implements relationship UsedIn
	InventoryItem	M:1	Implements relationship UsedIn
Stock	Warehouse	M:1	Implements relationship Stores
	InventoryItem	M:1	Implements relationship Stores
ItemStruct	InventoryItem	M:1	Implements relationship Contains
	InventoryItem	M:1	Implements relationship Contains
Employee	Project	M:1	Implements relationship Assigned
	Department	M:1	Implements relationship Belongs
Warehouse	Location	1:1	Implements relationship W-Situated
Department	Location	1:1	Implements relationship D◀Situated
EmployeePersonalInfo	Employee	M:1	Implements relationship ComposedOf
EmployeeEmploymentHistory	Employee	M:1	Implements relationship ComposedOf
EmployeeAcademicLog	Employee	M:1	Implements relationship ComposedOf
EmployeePublications	Employee	M:1	Implements relationship ComposedOf
EmployeeExtraCurricular	Employee	M:1	Implements relationship ComposedOf

Figure 3-14b. *Refined Relationships List for a Manufacturing Firm's Database*

3.7 Non-Relational Approaches

Prior to development of the relational model, the following approaches used to be employed:

- Inverted List Approach: Exemplified by DATACOM/DB
- Hierarchical Approach: Exemplified by Information Management System (IMS) database and the Random Access Management Information System (RAMIS)
- Network Approach: Exemplified by Conference on Data Systems Languages (CODASYL) initiative and Integrated Database Management System (IDMS) initiative

The strengths of the relational approach when compared to these approaches are its sound mathematical base, its flexibility, robustness, and simplicity.

In recent times, the *object oriented* (OO) model has been challenging the relational model on performance and efficiency for certain scenarios. Nonetheless, we expect that the two technologies will continue to peaceably coexist; huge investments have been made in relational database systems, and it is not likely that these will be abandoned. What is more likely to happen is that systems will be built based on relational databases, with object oriented user interfaces superimposed.

3.8 Summary and Concluding Remarks

Let us summarize what we have covered in this very important chapter:

- The relational database model is based on a number of fundamental concepts relating to the following: entity, entity set, relation, relationship, tuple, candidate key, primary key, alternate key, foreign key, domain, cardinality, degree.
- A domain is a named set of scalar values from which attribute values are drawn.
- A relation consists of a heading and a body. The heading consists of atomic attributes defined on specific domains. The body consists of a set of attribute-values pairs, where each attribute has a value drawn from its domain.
- In a database system, you are likely to find any combination of the following types of relations: base relations, logical views, snapshots, query results, intermediate results, and temporary relations.
- A relational database system (RDBS) is a collection of time-varying normalized relations, managed through an appropriate user interface, and with desirable constraints and features that enhance the effective, efficient management of the database.

- A relationship is an inherent mapping involving two or more relations. There are six types of relationships: one-to-one (1:1) relationship, one-to-many (1:M) relationship, many-to-one (M:1) relationship, many-to-many (M:M) relationship, component relationship, and subtype relationship.
- An E-R diagram (ERD) is a graphical representation of a database model. It is important to know how to represent relations/entities and relationships on the ERD.
- It is important to know how to implement the various types of relationships in the actual database design.
- The relation-attributes list (RAL) and relationship list (RL) are two useful alternatives to the E-R diagram, especially for large, complex systems.
- Database approaches that preexisted the relational approach include the inverted-list approach, the hierarchical approach and the network approach.
- A contemporary alternative to the relational approach is the object-oriented approach. However, given the efficacy of both approaches, it is more likely that they will complement each other in the future, rather than compete against each other.

Take the time to go over this chapter more than once if you need to, and make sure that you are comfortable with the concepts covered. In the upcoming chapter, we will build on the information covered in this chapter, as we discuss integrity rules and normalization. These two topics form the foundation for the rest of the course.