The composite attribute $(p_1, p_2, \ldots, p_k)$ forms the primary key of the relationship **R**. An instance of the relationship **R** is represented by concatenating its attributes $(r_1, r_2, \ldots, r_m)$ with the primary keys of the instances of the entities involved in the relationship. Figure 2.27 represents such a relationship.

2.4.3 Representation of Entities

Consider an application such as a hotel and its restaurants. Here we use a simplified version of the strong entity set EMPLOYEE with the following attributes: ***Empl_No***, *Name*, *Skill*. The primary key for this entity is ***Empl_No***.

The entity set EMPLOYEE can be described as follows:

entity set EMPLOYEE
Empl_No: **numeric;** (* primary key*)
Name: **string;**
Skill: **string;**

We represent the entity set EMPLOYEE by a table that can, for the sake of simplicity, be named EMPLOYEE. This table contains a column for each of its attributes and a row for each instance of the entity. We add a new instance of the entity EMPLOYEE by adding a row to this table. We also delete or modify rows to reflect changes that occur when employees leave or upgrade their skills. Figure 2.24 depicts an EMPLOYEE table. (We assume that each employee has but one skill.)

The weak entity DEPENDENTS, having as before the attributes *Dependent_Name* and *Kinship_to_Employee*, is dependent on the strong entity EMPLOYEE. We represent the weak entity by the table DEPENDENTS, which contains a column for the primary key of the strong entity EMPLOYEE. The DEPENDENTS table in Figure 2.25 includes instances of the weak entities (Rick, spouse) and (Chloe, daughter), which are dependent on EMPLOYEE 123459.

In general, to represent a weak entity such as W with the attributes $w_1, w_2, w_3, \ldots, w_n$ such that the weak entity is dependent on strong entity S with the primary key $s_1, s_2, \ldots, s_p$, we use a table with a column for each of the above attributes.

Figure 2.24 The EMPLOYEE table.

EMPLOYEE

Empl_No	*Name*	*Skill*
123456	Ron	waiter
123457	Jon	bartender
123458	Don	busboy
123459	Pam	hostess
123460	Pat	bellboy
123461	Ian	maître d'

Figure 2.25 The DEPENDENTS table.

DEPENDENTS

Empl_No	*Name*	*Kinship_to_Employee*
123459	Rick	spouse
123459	Chloe	daughter
123458	Cathy	spouse

2.4.4 Representation of Relationship Set

The entity-relationship diagrams are useful in representing the relationships among entities. They show the logical model of the database. In Figure 2.26, an E-R diagram shows the relationship between the entity sets EMPLOYEE and POSITION. The relationship set is called *DUTY_ALLOCATION* and its attributes are *Date* and *Shift*.

A relationship set involving entity sets $E_1, E_2, \ldots, E_n$ can be represented via a record containing the primary key of each of the entities E_i and the attributes of the relationship. Where the relationship has no attributes, only the primary keys of the entity involved are used to represent the relationship set.

Data for an E-R relationship could be represented by a number of tables. Each of the entities involved in the relationship is represented by a table, as is the relationship among these entities. The relationship *DUTY_ALLOCATION* between the entities EMPLOYEE and POSITION, shown in Figure 2.26, is represented by three tables displayed in Figure 2.27.

The table EMPLOYEE contains data about the entities representing the hotel employees. POSITION contains data on the duties to be performed by the hotel's employees in the restaurants run by the hotel. A relationship set is also represented by a table. *DUTY_ALLOCATION* is represented by the table DUTY_ALLOCATION, which contains the primary keys of the entities EMPLOYEE and POSITION along with the attributes of the relationship *Date* and *Shift*.

Figure 2.26 E-R diagram showing *DUTY_ALLOCATION* relationship between entity sets EMPLOYEE and POSITION.

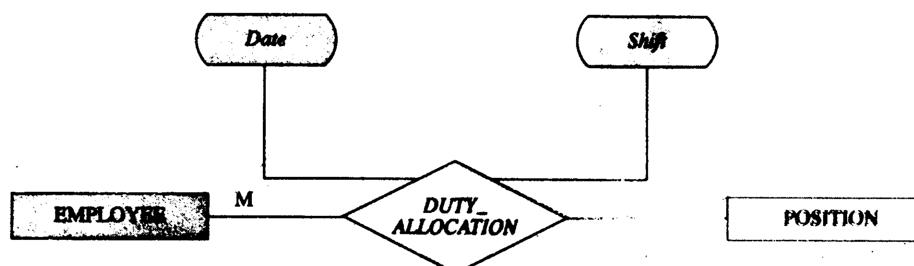
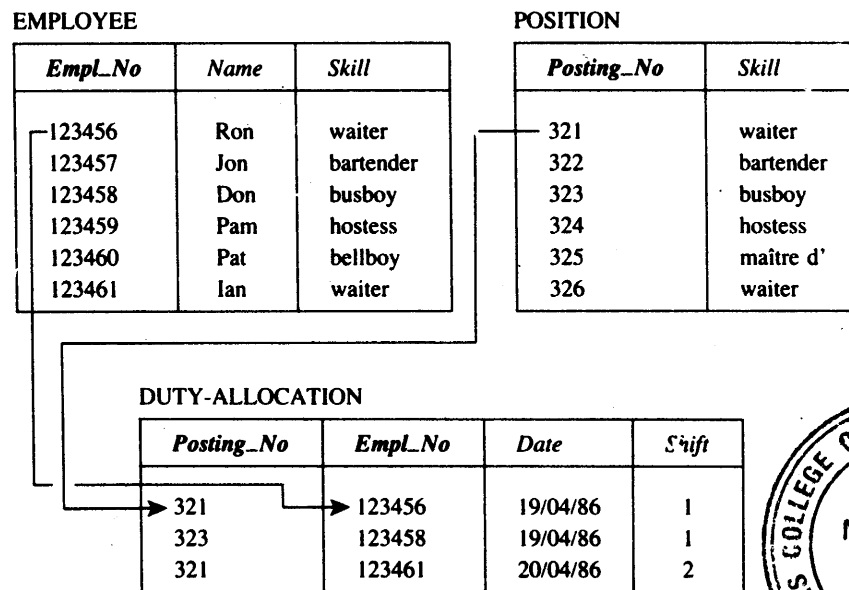


Figure 2.27 Representation of a relationship.



EMPLOYEE

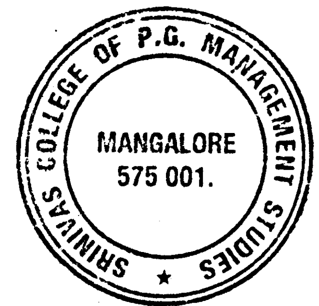
Empl_No	*Name*	*Skill*
123456	Ron	waiter
123457	Jon	bartender
123458	Don	busboy
123459	Pam	hostess
123460	Pat	bellboy
123461	Ian	waiter

POSITION

Posting_No	*Skill*
321	waiter
322	bartender
323	busboy
324	hostess
325	maître d'
326	waiter

DUTY-ALLOCATION

Posting_No	*Empl_No*	*Date*	*Shift*
321	123456	19/04/86	1
323	123458	19/04/86	1
321	123461	20/04/86	2



2.4.5 Generalization and Aggregation

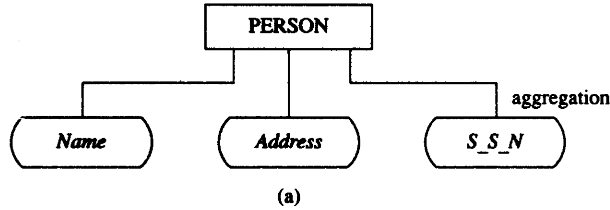
Abstraction is the simplification mechanism used to hide superfluous details of a set of objects, it allows one to concentrate on the properties that are of interest to the application. As such, car is an abstraction of a personal transportation vehicle but does not reveal details about model, year, color, and so on. Vehicle itself is an abstraction that includes the types car, truck, and bus.

There are two main abstraction mechanisms used to model information: generalization and aggregation. **Generalization** is the abstracting process of viewing sets of objects as a single general class by concentrating on the general characteristics of the constituent sets while suppressing or ignoring their differences. It is the union of a number of lower-level entity types for the purpose of producing a higher-level entity type. For instance, student is a generalization of graduate or undergraduate, full-time or part-time students. Similarly, employee is a generalization of the classes of objects cook, waiter, cashier, maître d'. Generalization is an *IS_A* relationship; therefore, manager *IS_An* employee, cook *IS_An* employee, waiter *IS_An* employee, and so forth. **Specialization** is the abstracting process of introducing new characteristics to an existing class of objects to create one or more new classes of objects. This involves taking a higher-level entity and, using additional characteristics, generating lower-level entities. The lower-level entities also inherit the characteristics of the higher-level entity. In applying the characteristic *size* to car we can create a full-size, mid-size, compact, or subcompact car. Specialization may be seen as the reverse process of generalization: additional specific properties are introduced at a lower level in a hierarchy of objects. Both processes are illustrated in Figure 2.28 wherein the lower levels of the hierarchy are disjoint.

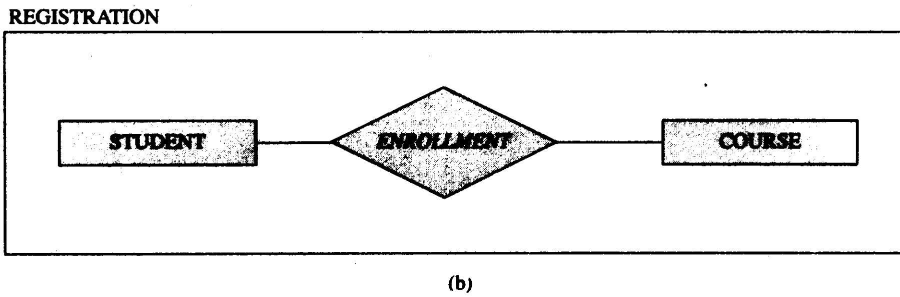
Aggregation is the process of compiling information on an object, thereby abstracting a higher-level object. In this manner, the entity person is derived by aggregating the characteristics name, address, and Social Security number. Another form of aggregation is abstracting a relationship between objects and viewing the relationship as an object. As such, the *ENROLLMENT* relationship between entities student and course could be viewed as entity REGISTRATION. Examples of aggregations are shown in Figure 2.31.

Consider the ternary relationship *COMPUTING* of Figure 2.23. Here we have a relationship among the entities STUDENT, COURSE, and COMPUTING SYSTEM.

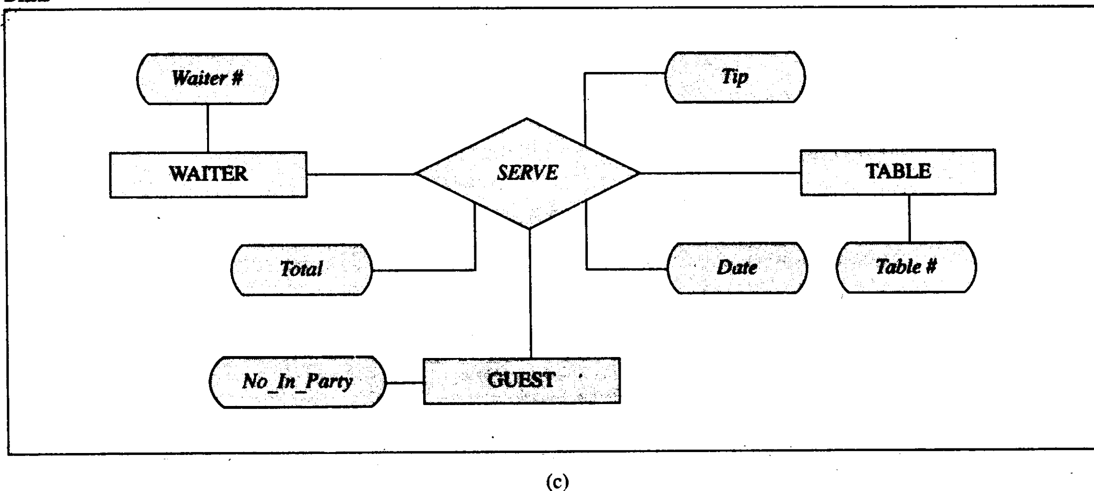
Figure 2.31 Examples of aggregation.



(a)



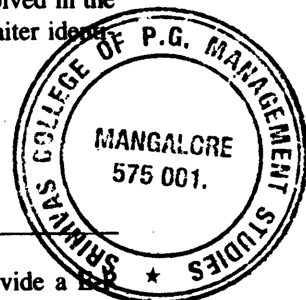
(b)



(c)

A student registered in a given course uses one of several computing systems to complete assignments and projects. The relationship between the entities STUDENT and COURSE could be the aggregated entity REGISTRATION (Figure 2.31b), as discussed above. In this case, we could view the ternary relationship of Figure 2.23 as one between registration and the entity computing system. Another method of aggregating is to consider a relationship consisting of the entity COMPUTING SYSTEMs being assigned to COURSEs. This relationship can be aggregated as a new entity and a relationship established between it and STUDENT. Note that the difference between a relationship involving an aggregation and one with the three entities lies in the number of relationships. In the former case we have two relationships; in the latter, only one exists. The approach to be taken depends on what we want to express. We would use the ternary relationship to express the fact that a STUDENT or COURSE cannot be independently related to a COMPUTING SYSTEM.

Let us investigate the relationship among the entities WAITER, TABLE, and GUEST shown in Figure 2.31c. These entities are of concern to a restaurant. There is a relationship, *SERVE,* among these entities; i.e., a waiter is assigned to serve guests at a given table. The waiters could be assigned unique identifiers. For example, a waiter is an employee and the employee number uniquely identifies an employee and hence a waiter. A table could be assigned a number; however, this may be more informal, since on occasion two or more tables are put together to accommodate a group of guests. The guests, even though identifiable by their features and other unique identifiers such as *Social_Security_Number* or driver's license number, are not distinguishable for this application. Thus the *SERVE* relationship can best be handled by an aggregation. The aggregation can be called a BILL (Figure 2.31c), and requires an introduction of an unique bill number for identification. In addition, the following attributes from the SERVE relationship and the entities involved in the relationship can be used for the aggregated entity: unique bill number, waiter identifier, table identifier, date, number of guests in party, total, tip.



2.5 A Comparative Example

In this section we describe a small database modeling problem and provide a E-R model for it. We give its implementation in each of the other three modeling schemes in Sections 2.6, 2.7, and 2.8.

Consider a database for the Universal Hockey League (UHL), a professional ice hockey league with teams worldwide. It consists of a number of divisions and numerous franchises under each division. The database records statistics on teams, players, and divisions of the league.

A franchise may relocate to another city and may become part of a different division. Players are under contract to a franchise and are obliged to move with it. This relationship between a franchise and a division is called a team. We use the word team synonymously with franchise. Consequently, we can view a franchise as consisting of a collection of players, coaches, and a general manager. Players are required to play for a given franchise for the entire season.

First we present the entity relationship diagram. We convert the E-R diagram to relational, network, and hierarchical models in Sections 2.6, 2.7, and 2.8.

Figure 2.33 A tabular representation of relations.

EMPLOYEE

Empl_No	*Name*	*Skill*
123456	Ron	waiter
123457	Jon	bartender
123458	Don	busboy
123459	Pam	hostess
123460	Pat	bellboy
123461	Ian	waiter

POSITION

Posting_No	*Skill*
321	waiter
322	bartender
323	busboy
324	hostess
325	maître d'
326	waiter

been partitioned into the various relations. While it is possible to infer access paths from the relational model, as we will see later, the relational approach does require the user to provide logical navigation through the database for the query.

The relation is the only data structure used in the relational data model to represent both entities and the relationships between them. A relation may be visualized as a named table. Figure 2.33 shows the two relations EMPLOYEE and POSITION using a tabular structure. Each column of the table corresponds to an attribute of the relation and is named.

Rows of the relation are referred to as **tuples** of the relation and the columns are its **attributes.** Each attribute of a relation has a distinct name. The values for an attribute or a column are drawn from a set of values known as a **domain.** The domain of an attribute contains the set of values that the attribute may assume. In the relational model, note that no two rows of a relation are identical and the ordering of the rows is not significant.

A relation represented by a table having n columns, defined on the domains D_1, D_2, . . . , D_n is a subset of the cartesian product $D_1 \times D_2 \times \ldots D_n$.

A relationship is represented, as in the E-R model, by combining the primary keys of the entities involved in a relation and its attributes, if any.

A correspondence between two relations is implied by the data values of attributes in the relation defined on common domains. Such correspondence is used in navigating through the relational database. In the example in Figure 2.33 both the EMPLOYEE and POSITION relations contain the identically named attribute[1] *Skill* defined on a common domain. Consequently we can join these two relations to form the relation, POSITION_ELIGIBILITY (Figure 2.34) using the common values of the attribute *Skill*. Joining the two relations involves taking two rows, one from each table, such that the value of *Skill* in the two rows is identical, and then concatenating these rows. Note that in Figure 2.34 the first attribute *Skill* is from the EMPLOYEE relation and the second is from the POSITION relation. Qualifying these attributes in POSITION_ELIGIBILITY by their respective relation names would allow us to more strictly adhere to the relational model where names of attributes in the same relation are distinct.

[1]The names of these attributes are identical in this instance to remind us that they have a common domain.

Figure 2.34 The relation obtained after joining the two relations of Figure 2.33

POSITION_ELIGIBILITY

Empl_No	*Name*	EMPLOYEE. *Skill*	*Posting_No*	POSITION. *Skill*
123456	Ron	waiter	321	waiter
123456	Ron	waiter	326	waiter
123457	Jon	bartender	322	bartender
123458	Don	busboy	323	busboy
123459	Pam	hostess	324	hostess
123461	Ian	waiter	321	waiter
123461	Ian	waiter	326	waiter

Relational Model for the UHL

Using the relational model, each of the entities in the UHL can be represented by a relation. The description of the relation is given by a **relation scheme.** A relation scheme is like a type declaration in a programming language. It indicates the attributes included in the scheme, their order, and their domain. However, we will ignore the domain for the present.

Each relation scheme is named and we indicate this name by boldface capital letters. We have a relation scheme for each of the PLAYER, FRANCHISE, and DIVISION relations. These relation schemes are similar to the corresponding entities in the E-R model:

PLAYER (***Name***, *Birth_Place, Birth_Date*)
FRANCHISE (***Franchise_Name***, *Year_Established*)
DIVISION (***Division_Name***)

Relationships between entities are also represented by relations.

The relationship *GOAL* is represented by a relation whose scheme includes the primary keys *Name* and *Franchise_Name*, respectively, of the entities PLAYER and FRANCHISE. In addition, it contains the attributes corresponding to those of the relationship, namely *Year, Goals_Against_Avg*, and *Shutouts*. Therefore, the relation scheme for GOAL is:

GOAL(***Name, Franchise_Name***, *Year, Goals_Against_Avg, Shutouts*)

FORWARD is also represented by a relation scheme with attributes that consist of the same primary keys *Name* and *Franchise_Name*. It contains, as well the attributes *Year, Goals* and *Assists*. Accordingly, the relation scheme for FORWARD is:

FORWARD (***Name, Franchise_Name***, *Year, goals, Assists*)

TEAM is represented by a relation scheme with attributes consisting of the primary keys *Franchise_Name* and *Division_Name*, respectively, of the entities FRANCHISE and DIVISION. It also contains the attributes corresponding to those of the relationship, namely *Year, City*, and *Points*. The relation scheme for TEAM is:

TEAM (***Franchise_Name, Division_Name***, *Year, City, Points*)

Figure 2.35 Parts of relations from the UHL relation database.

PLAYER

Name	*Birth_Place*	*Birth_Date*
Zax Viviteer	Prague, Czec	1962-04-29
Barn Kurri	Detroit, Mich	1964-07-17
Todd Smith	Roseau, Minn	1963-05-09
Dave Fisher	Edmonton, Canada	1959-10-28
Ozzy Xavier	Kiruna, Sweden	1965-02-19
Gaston Vabr	Montreal, Canada	1958-05-12
Ken Dorky	Chicago, Ill	1958-05-13
Brian Lafontaine	Paris, France	1960-07-03
Bruce McTavish	Rio, Brazil	1966-10-27
Dave O'Connell	Dublin, Ireland	1967-03-16
Johnny Brent	Boston, Mass	1964-12-23

FRANCHISE

Franchise_Name	*Year_Established*
Bullets	1975
Rodeos	1921
Zippers	1917
Blades	1982
Flashers	1967

DIVISION

Division_Name
Northern
Southern
European
World

FORWARD

Name	*Franchise_Name*	*Year*	*Goals*	*Assists*
Barn Kurri	Bullets	1986	40	67
Bruce McTavish	Bullets	1986	30	37
Todd Smith	Rodeos	1986	17	24
Ozzy Xavier	Blades	1986	56	119
Ozzy Xavier	Flashers	1985	36	49
Gaston Vabr	Flashers	1986	16	22
Zax Viviteer	Blades	1986	80	162
Dave O'Connell	Zippers	1986	12	59
Brian Lafontaine	Zippers	1985	10	40
Brian Lafontaine	Zippers	1986	22	73

Sample tuples from these relations, which have the same names as the corresponding schemes, are shown in the tables of Figure 2.35.

We return to in-depth discussions of the relational data model in Chapter 4.

Figure 2.35 Continued

GOAL

Name	*Franchise_Name*	*Year*	*Goals_Against_Avg*	*Shutouts*
Ken Dorky	Blades	1986	1.21	7
Dave Fisher	Zippers	1986	4.02	4
Johnny Brent	Flashers	1986	7.61	0
Dave Fisher	Flashers	1985	3.05	5

TEAM

Franchise_Name	*Division_Name*	*Year*	*City*	*Points*
Flashers	Northern	1986	St. Louis	93
Blades	Northern	1986	Edmonton	97
Zippers	European	1985	Paris	82
Zippers	Northern	1986	Montreal	99
Rodeos	Southern	1986	Rio	65
Bullets	World	1986	Tokyo	79

2.7 Network Data Model

The network data model was formalized in the late 1960s by the Database Task Group of the Conference on Data System Languages (DBTG/CODASYL). Their first report (CODA 71), which has been revised a number of times, contained detailed specifications for the network data model (a model conforming to these specifications is also known as the DBTG data model). The specifications contained in the report and its subsequent revisions have been subjected to much debate and criticism. Many of the current database applications have been built on commercial DBMS systems using the DBTG model.

The DBTG model uses two different data structures to represent the database entities and relationships between the entities, namely record type and set type. A **record type** is used to represent an entity type. It is made up of a number of data items that represent the attributes of the entity.

A **set type** is used to represent a directed relationship between two record types, the so-called **owner record type,** and the **member record type.** The set type, like the record type, is named and specifies that there is a one-to-many relationship (1:M) between the owner and member record types. The set type can have more than one record type as its member, but only one record type is allowed to be the owner in a given set type. A database could have one or more occurrences of each of its record and set types. An occurrence of a set type consists of an occurrence of the owner record type and any number of occurrences of each of its member record types. A record type cannot be a member of two distinct occurrences of the same set type.

the member record type of each of these symmetrical sets. Corresponding to the relationship *GOAL*, we create the logical record type GOAL, and the sets *Fr_G* and *P_G*. The record types FRANCHISE and PLAYER are owners and the record type GOAL is the common member in these sets.

The data structure diagram for the database for the UHL is shown in Figure 2.38. The sets included are:

- *P_G* and *Fr_G*, corresponding to the many-to-many relationship *GOAL* between the entities PLAYERS and FRANCHISE. GOAL is the common member record type, the owner record types being PLAYER (of the set *P_G*) and FRANCHISE (of the set *Fr_G*). The attributes of the relationship are the fields of the record type GOAL.
- *P_F* and *Fr_F*, corresponding to the many-to-many relationship *FORWARD* between the entities PLAYERS and FRANCHISE. The member record type is FORWARD, with PLAYER (of the set *P_F*) and FRANCHISE (of the set *Fr_F*) being the owner record types. The fields of the common member record type FORWARD are the attributes of the relationship.
- *Fr_T* and *D_T*, corresponding to the many-to-many relationship *TEAM* between the entities FRANCHISE and DIVISION. TEAM is the member record type; the owner record types are FRANCHISE (of the set *Fr_T*) and DIVISION (of the set *D_T*). The attributes of the relationship are the fields of the record type TEAM, the common member of *Fr_T* and *D_T*.

Figure 2.39 features a sample of the data contained in some of these logical record types and some of the sets in which these records are involved as member or owner. The common records, which are shaded, are the links in establishing a many-to-many relationship. The connecting lines between two records indicate the exis-

Figure 2.38 Network model for the UHL database.

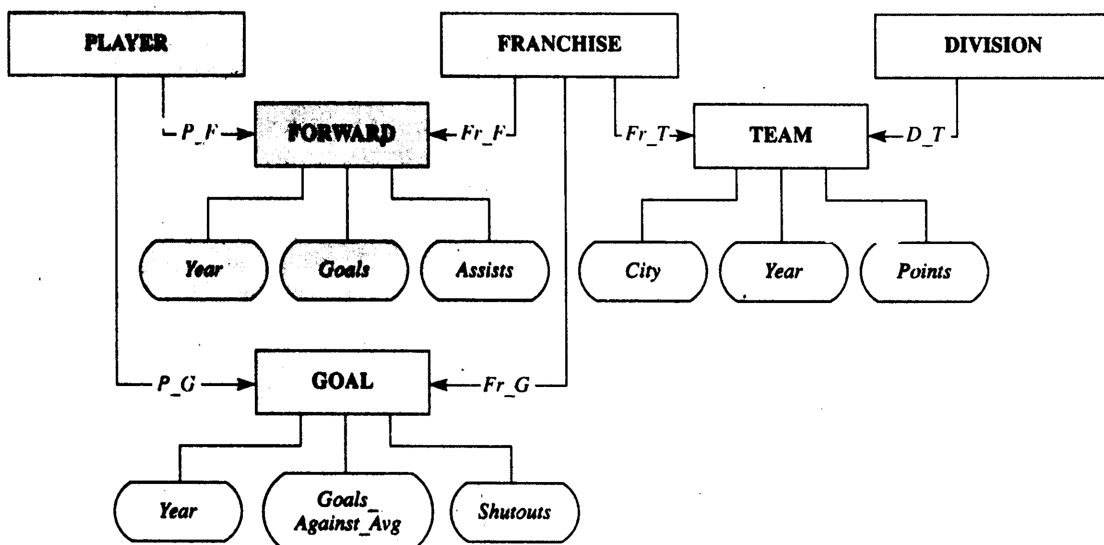
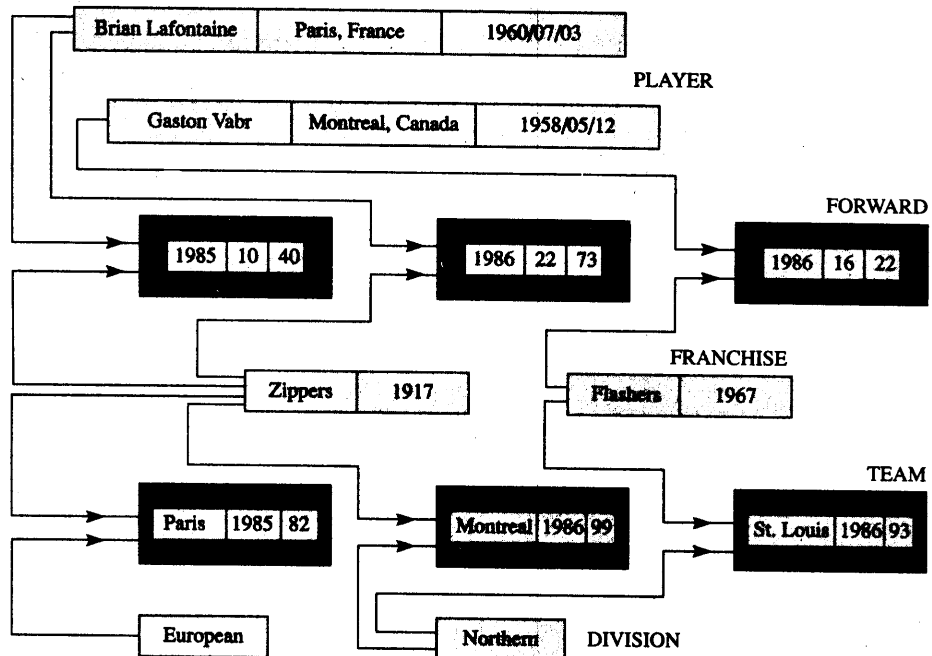


Figure 2.39 Part of the data in the network database of the UHL.



tence of an owner/member relationship between the record occurrences and some mechanism to go from one to the other. For instance, the occurrence (Brian Lafontaine, Paris, France, 1960-07-03) of the logical record type PLAYER is the owner in the set occurrence *P_F*. The members of this set occurrence owned by him are the FORWARD logical record occurrences (1985, 10, 40) and (1986, 22, 73). These are also owned by the franchise Zippers and establish the relationship between the player and the franchise.

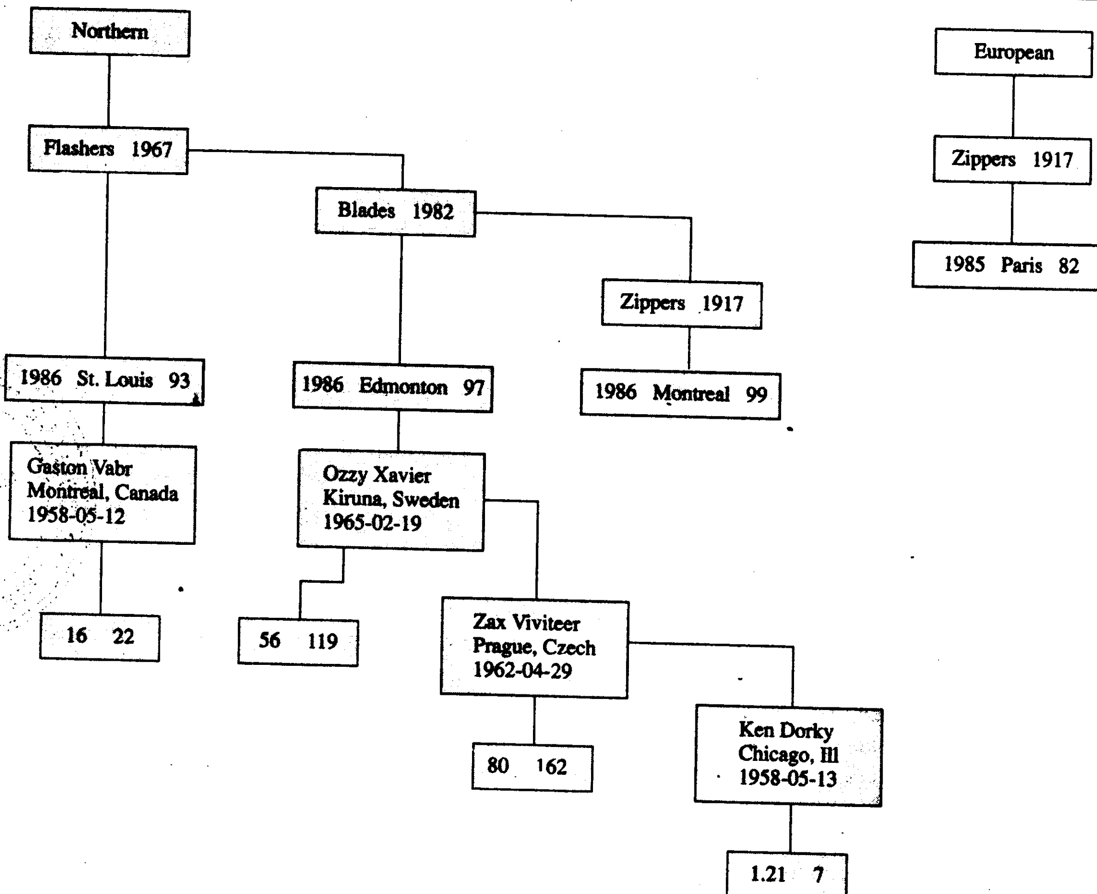
We return to detailed discussions of the network model in Chapter 8

2.8 Hierarchical Model

A tree may be defined as a set of nodes such that there is one specially designated node called the root (node) and the remaining nodes are partitioned into disjoint sets, each of which in turn is a tree, the subtrees of the root. If the relative order of the subtrees is significant, the tree is an ordered tree.

Like an organization chart or a family tree, a hierarchy is an ordered tree and is easy to understand. At the root of the tree is the single parent; the parent can have none, one, or more children. (Note that in comparing the hierarchical tree with a family tree, we are ignoring one of the parents; in other words, both the parents are represented implicitly by the single parent.)

Figure 2.41 Part of the data in the hierarchical database of the UHL.

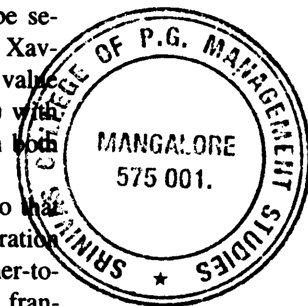


of the entity is represented by a record occurrence. A weak entity can be represented as a separate record type. In this case, the identifying relationship is represented as a set type wherein the record type corresponding to the weak entity type forms the member and the record type corresponding to the strong entity is the owner. A 1:1 or 1:N relationship is represented as a set type. An M:N relationship requires introducing an **intermediate record type.** This record type is a common member in two set types, one of which is owned by each of the record types involved in the M:N relationship.

Converting an E-R diagram to a hierarchical model can be accomplished as follows. Each entity type in the E-R diagram is represented by a record type. A 1:N relationship is represented as a hierarchy type where the record type corresponding to the one side of the relationship is the parent (a 1:1 relationship is a special case of the 1:N relationship). A weak entity can be represented as a separate record type. This record type becomes a dependent in a hierarchy where the record type, corresponding to the strong entity in the identifying relationship, is the parent. An M:N relationship requires introducing duplicate record types or using multiple hierarchies and introducing virtual records.

In the network model, it is possible that several identical occurrences of the same logical record type could exist. These multiple identical occurrences are distinguished by their membership in different occurrences of the sets. Similarly, in the hierarchical model, identical occurrences of a record type are distinguished by their associations with different ancestor record type occurrences. The tuples of a relation are, however, unique because if the relation represents a relationship between entities, the relationships between occurrences of the entities are explicitly recorded in the tuples by inclusion of the corresponding primary keys.

The relational model allows for a fairly straightforward method of **selecting** certain entities or relationships. This is done by selecting those tuples of the relation corresponding to the entity or relationship that meet certain selection conditions. For instance, all franchises for which the player Ozzy Xavier played could be derived by choosing tuples from the relation FORWARD (Figure 2.35) with *Name* = Ozzy Xavier. Similarly, all players who scored more than 50 goals in 1986 could be selected from the FORWARD relation. Likewise, finding all cities in which Ozzy Xavier played can be done by first selecting the tuples from FORWARD with the value of *Name* = Ozzy Xavier. These selected tuples are then joined (concatenated) with those in the table TEAM such that the values of *Franchise_Name* and *Year* in both is the same.



In the network model, the selection operation on a record type is similar to that in the relational model. However, the operation corresponding to the join operation of the relational model is handled differently. This involves following the owner-to-member or the member-to-owner pointers. Therefore, in order to identify all franchises for which Ozzy Xavier played, we would first find the record for Ozzy Xavier in the player record type. We would then follow the pointers in the set *P_F* to the occurrences of the member record type FORWARD for his score, and last, follow the pointers to the owner of each such occurrence in the set *Fr_F* to find the FRANCHISE. Since the player Ozzy Xavier is not a goalie, the set *P_G* for the occurrence of his record in record type PLAYER would be empty. Consequently, following the set *P_G* and then the owner in the set *Fr_G* for this player would not be possible.

Selection operations for the record type corresponding to the root of a hierarchical tree are similar to operations for its counterpart in the relational and network models. As in the case of the network model, we have to traverse pointers from parent to child since there is no method of traveling from descendant to parent. However, a virtual scheme using a virtual record concept (to be discussed in chapter 9) introduces this reverse-navigation facility.

The process of joining relations in the case of the relational model or following the pointers from owner to member, from member to owner, or from parent to child is known as **navigating** through the database. Navigation through relations that represent an M:N relationship is just as simple as through a 1:M relationship. This leads us to conclude that it is easier to specify how to manipulate a relational database than a network or hierarchical one. This in turn leads to a query language for the relational model that is correct, clear, and effective in specifying the required operations. Unfortunately, the join operation is inherently inefficient and demands a considerable amount of processing and retrieval of unnecessary data. The structure for the network and hierarchical models can be implemented efficiently. Such an implementation would mean that navigating through these databases, though awkward, requires the retrieval of relatively little unnecessary data.

2.10 Summary

In this chapter we previewed the major data modeling concepts and the data models used in current DBMSs. The E-R model is used increasingly as a tool for database applications modeling.

A number of data representation models have been developed over the years. As in the case of programming languages, one concludes that there is no one "best" choice for all applications. These models differ in their method of representing the associations between entities and attributes.

Traditional data models are hierarchical, network, or relational models. The hierarchical model evolved from the file-based system; the network model is a superset of the hierarchical model. The relational data model is based on the mathematical relational concept. The data model concept evolved at about the same time as the relational data model.

The entity-relationship data model, which is popular for high-level database design, provides a means of representing relationships between entities. The entity relationship data model was developed using commercially available DBMSs to model application databases.

The DBTG proposal was the first data model to be formalized in the late 1960s. Many current database applications have been built on commercial DBMSs using this approach.

Key Terms

data model
association
attribute association
relationship
functional dependency
determinant
candidate key
primary key
binary relationship
repeating group
file-based model
hierarchical model
network model
relational data model
semantic data model
entity-relationship (E-R) data model
entity-relationship (E-R) diagram
strong entity
identifying relationship
weak entity
discriminator
relationship set
N-ary relationship
ternary relationship
abstraction
generalization
specialization
aggregation
tuple
attribute
domain
relation scheme
record type
set type
owner record type
member record type
logical record
forest
spanning trees
selecting
intermediate record type
navigating

Exercises

2.1 Define the following terms:

(a) association
(b) relationship