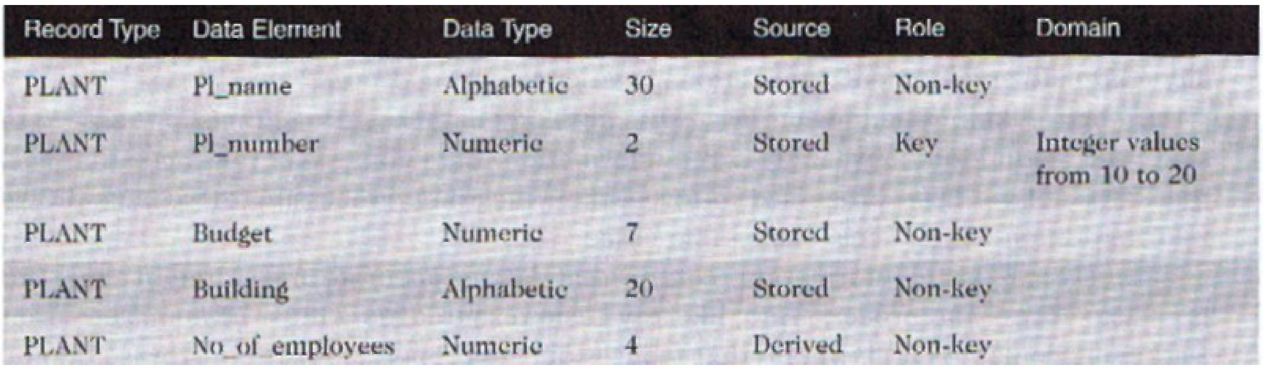
Q1. Explain Metadata using an example.

Metadata, in a database environment, is data that describes the properties of data. It contains a complete definition or description of database structure (i.e., the file structure, data type, and storage format of each data item), and other constraints on the stored data. For example, wheSn the structure of the ten digits 1713445232 is revealed, the ten digits become information, such as a phone number. Metadata defines this structure. In other words, through the lens of metadata, data takes on specific meaning and yields information.1 Table 1.1 contains metadata for the data associated with a manufacturing plant.

Table 1.1 Some metadata for a manufacturing plant



Record Type	Data Element	Data Type	Size	Source	Role	Domain
PLANT	Pl_name	Alphabetic	30	Stored	Non-key	
PLANT	Pl_number	Numeric	2	Stored	Key	Integer values from 10 to 20
PLANT	Budget	Numeric	7	Stored	Non-key	
PLANT	Building	Alphabetic	20	Stored	Non-key	
PLANT	No_of_employees	Numeric	4	Derived	Non-key	

Q2. Explain the limitations of file processing systems. How do database system makes it possible to overcome these limitations?

File-processing systems suffer from a number of limitations:

- Lack of data integrity. Data integrity ensures that data values are correct, consistent, complete, and current. Duplication of data in isolated file-processing systems leads to the possibility of inconsistent data. Then it is difficult to identify which of these duplicate data is correct, complete, and/or current. This creates data integrity problems.
- Lack of standards: Organizations with file-processing systems often lack or find it difficult to enforce standards for naming data items as well as for accessing, updating, and protecting data. The absence of such standards can lead to unauthorized access and accidental or intentional damage to or destruction of data. In essence, security and confidentiality of information may be compromised.
- Lack of flexibility/maintainability. Information systems make it possible for end users to develop information requirements that they had never envisioned previously. This inevitably leads to a substantial increase in requests for new queries and reports. However, file-processing systems arc dependent upon a programmer who cither has to write or modify program code to meet these information requirements from isolated data.

Database systems do not suffer from this limitation as data structures are stored, along with the data, in the database as opposed to being recorded in the application program. The DBMS accesses the database and supplies data appropriate to the needs of individual application programs. The DBMS accomplishes this using the data dictionary where the metadata is recorded. Thus the data dictionary in a database environment insulates application programs from changes to the structure of data. The notion of data relatability involves the creation of logical relationships between different types of records, usually in different files. In a file-processing environment, information often cannot be generated without a programmer writing or at least modifying an application program to consolidate the files. With the advent of database systems, all that is necessary is for one to specify the data to be combined; the DBMS will perform the necessary operations to accomplish the task.

Q3. Difference between:

a) Database and DBMS

Database	Database Management System (DBMS)
A **database** is a collection of related data.	A **DBMS** is a software system that manages databases.
It stores and organizes data in tables, rows, and columns.	A **DBMS** provides tools to create, manipulate, and manage databases.
A **database** contains raw data, which can be accessed and modified.	A **DBMS** enables users to interact with and manage data in the database using tools and query languages (like SQL).
It does not provide any tools for interacting with the data.	A **DBMS** provides functionalities like data insertion, retrieval, update, deletion, security, backup, and recovery.
A **database** is passive, just storing data.	A **DBMS** is active, providing mechanisms for data consistency, concurrency, and integrity.
It lacks an interface for querying and manipulating the data.	A **DBMS** includes query languages (such as SQL), data definition, and data manipulation tools.
A **database** may store the data along with its metadata.	A **DBMS** manages and controls the access to the database, ensuring multi-user access and proper resource management.
Examples: **MySQL database**, **Oracle database**	Examples: **Oracle DBMS**, **Microsoft SQL Server**, **MySQL DBMS**

b) Stored and Derived Attributes

Stored Attributes	Derived Attributes
Stored attributes are physically stored in the database.	**Derived attributes** are not stored directly but are derived from other attributes.
The values of **stored attributes** are explicitly entered or recorded in the database.	The values of **derived attributes** are computed or calculated based on other stored data.
Example: **Employee's salary** stored in the database.	Example: **Flight time**, which is calculated from the arrival time minus the departure time.
Stored attributes are permanent and remain consistent unless explicitly updated.	**Derived attributes** can change automatically if the values of the base attributes change.
Stored attributes must always have a value assigned to them (except if they are optional).	**Derived attributes** may not need to be stored since their values can be recalculated whenever needed.
Stored attributes are typically simpler to manage and query.	**Derived attributes** require calculation logic whenever they are queried.
Example: **Date of birth** of an employee.	Example: **Age**, which is derived from the Date of birth.

c) Key and Non Key Attributes

Key Attributes	Non-Key Attributes
Key attributes are the attributes that uniquely identify an entity in a database.	**Non-key attributes** are attributes that do not uniquely identify an entity.
They are part of the **unique identifier** for an entity type.	They are not part of the unique identifier and are used to store additional information about an entity.
Key attributes are mandatory and essential for the identity of an entity.	**Non-key attributes** may be optional and are not necessary for identifying an entity.
Example: **SSN (Social Security Number)**, **Pat_ID (Patient ID)** are key attributes.	Example: **Address**, **Phone Number**, **Age** are non-key attributes.
Key attributes are used to form primary keys or unique constraints in a database.	**Non-key attributes** are used to provide more descriptive details about the entity.
Key attributes are often underlined in ER diagrams to signify their importance in identification.	**Non-key attributes** are typically not underlined in ER diagrams.
Key attributes play a crucial role in indexing and ensuring data integrity in relationships.	**Non-key attributes** are mainly used to describe properties or characteristics of entities.

Q4. How does cardinality constraint and participation relates to notion of partial and total participation?

The data integrity constraints pertaining to relationship types specified in an ER diagram are referred to as the structural constraints of a relationship type.8 Two independent Structural constraints together define a relationship type: cardinality constraint (also known as mapping cardinality or connectivity) and participation constraint.

The cardinality constraint for a binary relationship type is a constraint that specifies the maximum number of entities of an entity type to which another entity can be associated through a specific relationship set expressed as a ratio.

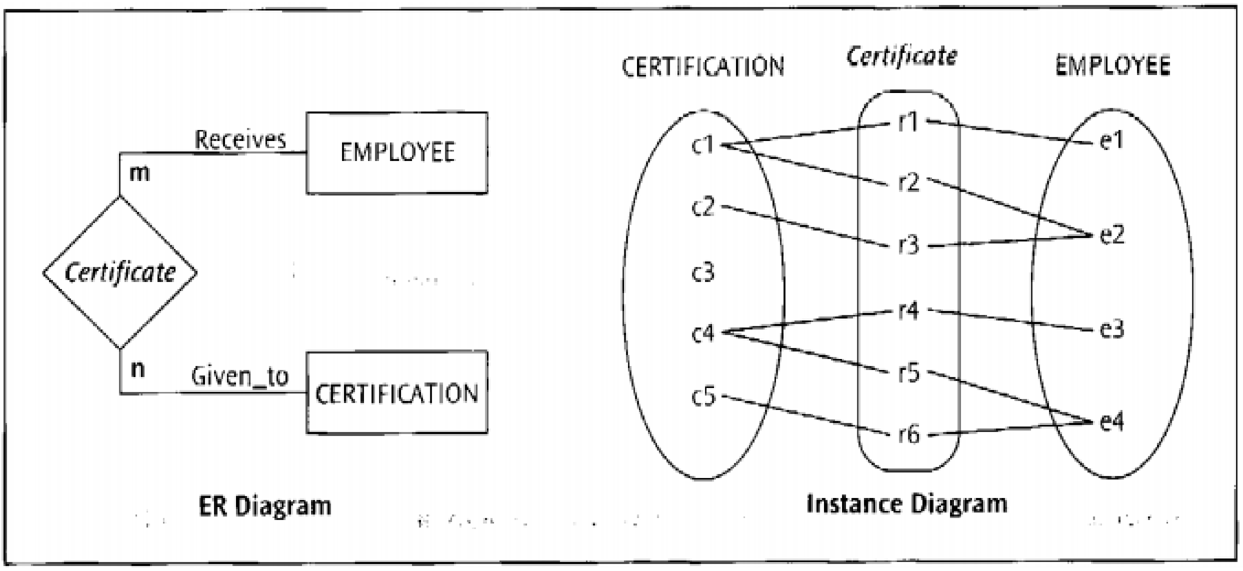


Figure 2.12 An illustration of cardinality ratio of m:n

The cardinality constraint reflects the maximum cardinality of the entity types participating in the binary relationship type. The maximum cardinality indicates the maximum number of relationship instances in which an entity participates.

The participation constraint for an entity type in a binary relationship type is based on whether, in order to exist, an entity of that entity type needs to be related to an entity of the other entity type through this relationship type. Participation can be total or partial. If, in order to exist, every entity must participate in the relationship, then participation of the entity type in that relationship type is total participation. On the other hand, if an entity can exist without participating in the relationship, then participation of the entity type in that relationship type is partial participation. Total and partial participation are also commonly referred to as mandatory and optional participation, respectively. For example, if every Salesperson must have sold at least one Vehicle, then there is total participation of SALESPERSON in the Sales relationship type. Instead, if a Salesperson need not have sold any Vehicle, then there is partial participation of a SALESPERSON in the Sales relationship type. Since the participation constraint specifies the minimum number of relationship instances in which each entity can participate, it reflects the minimum cardinality of an entity type's participation in a relationship type.

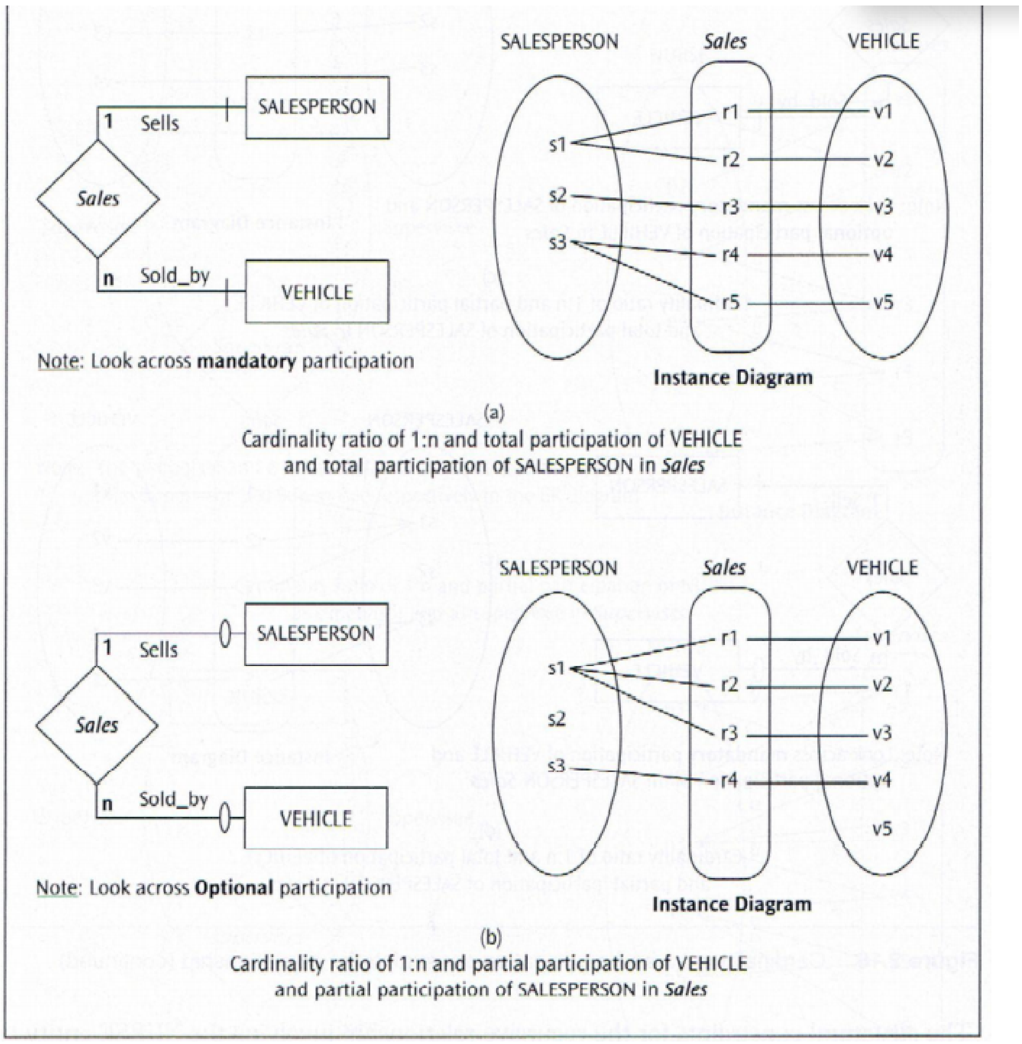

SALESPERSON
Sales
VEHICLE
1
Sells
n
Sold_by
s1
s2
s3
r1
r2
r3
r4
r5
v1
v2
v3
v4
v5
Note: Look across mandatory participation
Instance Diagram
(a)
Cardinality ratio of 1:n and total participation of VEHICLE and total participation of SALESPERSON in Sales
Note: Look across Optional participation
(b)
Cardinality ratio of 1:n and partial participation of VEHICLE and partial participation of SALESPERSON in Sales

Figure 2.16 Cardinality ratio and participation constraints for a relationship

Total participation of an entity type in a relationship type is also called existence dependency of that entity type in that relationship type. Accordingly, in Figure 2.15a SALESPERSON has existence dependency on the Sales relationship type, whereas in Figure 2.15b SALESPERSON does not have existence dependency on the Sales relationship type.

The structural constraints for the relationship between the SALESPERSON and VEHICLE appear in Figure 2.16. In each of Figures 2.16a through 2.16d, a SALESPERSON can sell many VEHICLES. In Figure 2.16a, a SALESPERSON must sell at least one VEHICLE and a VEHICLE must be sold by a SALESPERSON. In Figure 2.16b a SALESPERSON may or may

not sell a VEHICLE and a VEHICLE need not be sold by a SALESPERSON. Figure 2.16c illustrates the requirement that a SALESPERSON must sell at least one VEHICLE, while a VEHICLE need not be sold by a SALESPERSON. Finally, in Figure 2.16d a SALESPERSON may or may not sell a VEHICLE but a VEHICLE must be sold by a SALESPERSON.

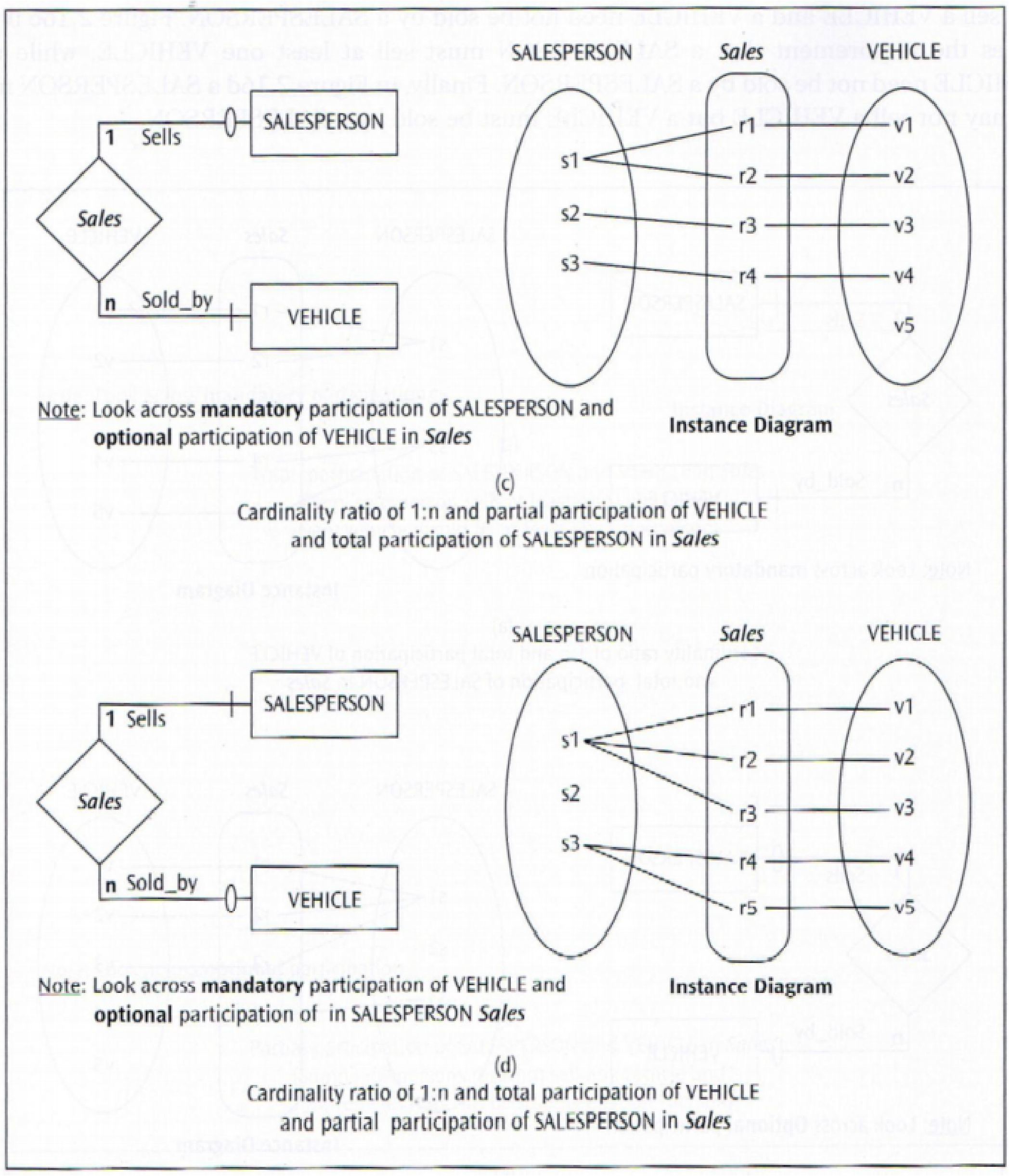

SALESPERSON
Sales
VEHICLE
1 Sells
SALESPERSON
Sales
n Sold_by
VEHICLE
s1
s2
s3
r1
r2
r3
r4
v1
v2
v3
v4
v5
Note: Look across **mandatory** participation of SALESPERSON and **optional** participation of VEHICLE in *Sales*
Instance Diagram
(c)
Cardinality ratio of 1:n and partial participation of VEHICLE and total participation of SALESPERSON in *Sales*
SALESPERSON
Sales
VEHICLE
1 Sells
SALESPERSON
Sales
n Sold_by
VEHICLE
s1
s2
s3
r1
r2
r3
r4
r5
v1
v2
v3
v4
v5
Note: Look across **mandatory** participation of VEHICLE and **optional** participation of in SALESPERSON *Sales*
Instance Diagram
(d)
Cardinality ratio of 1:n and total participation of VEHICLE and partial participation of SALESPERSON in *Sales*

Figure 2.16 Cardinality ratio and participation constraints for a relationship (continued)

Q5. What is meant by degree of relationship?

S

A relationship type is a meaningful association among entity types. The degree of a relationship type is defined as the number of entity types participating in that relationship type. A relationship type is said to be binary (or of degree two) when two entity types are involved. Relationship types that involve three entity types (of degree three) are defined as ternary relationships. While a relationship of degree four can be referred to as a quaternary relationship, a more generalized term beyond ternary relationship is n-ary relationship where the degree of the relationship type is n. An entity type related to itself is termed a recursive relationship type.

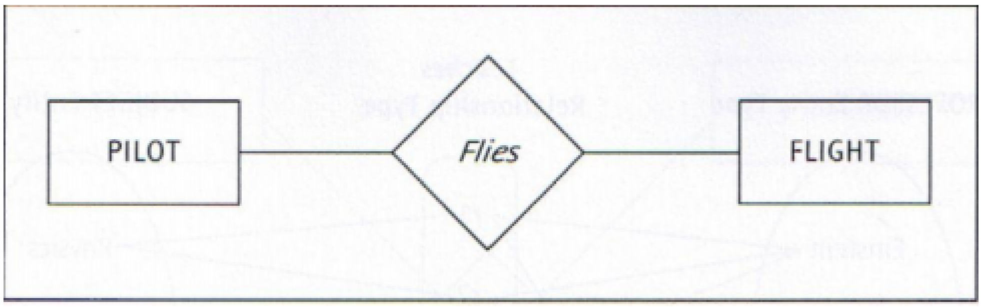


Figure 2.4 A binary relationship

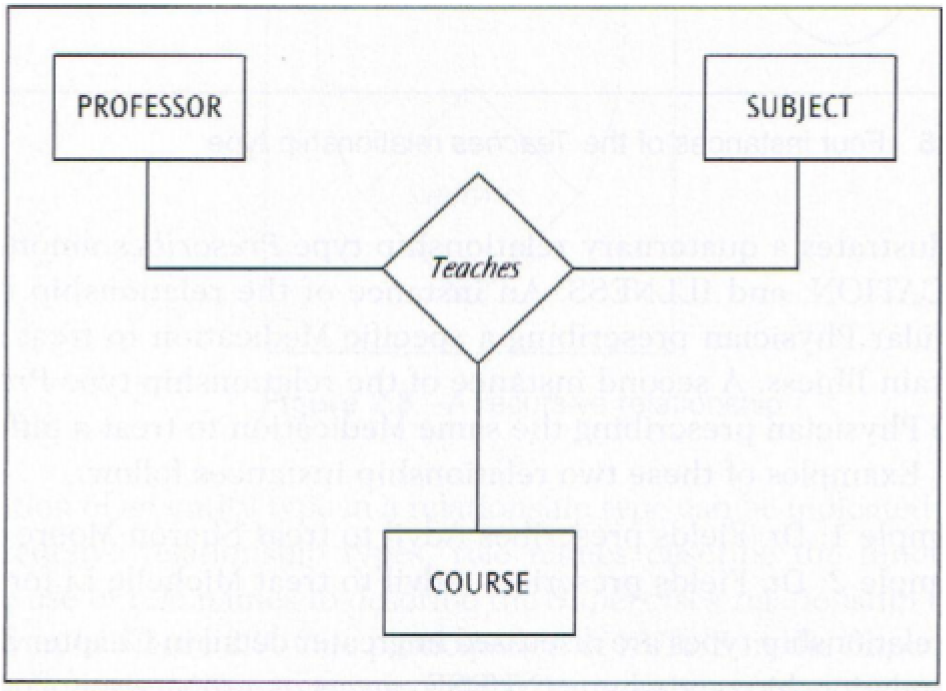


Figure 2.5 A ternary relationship

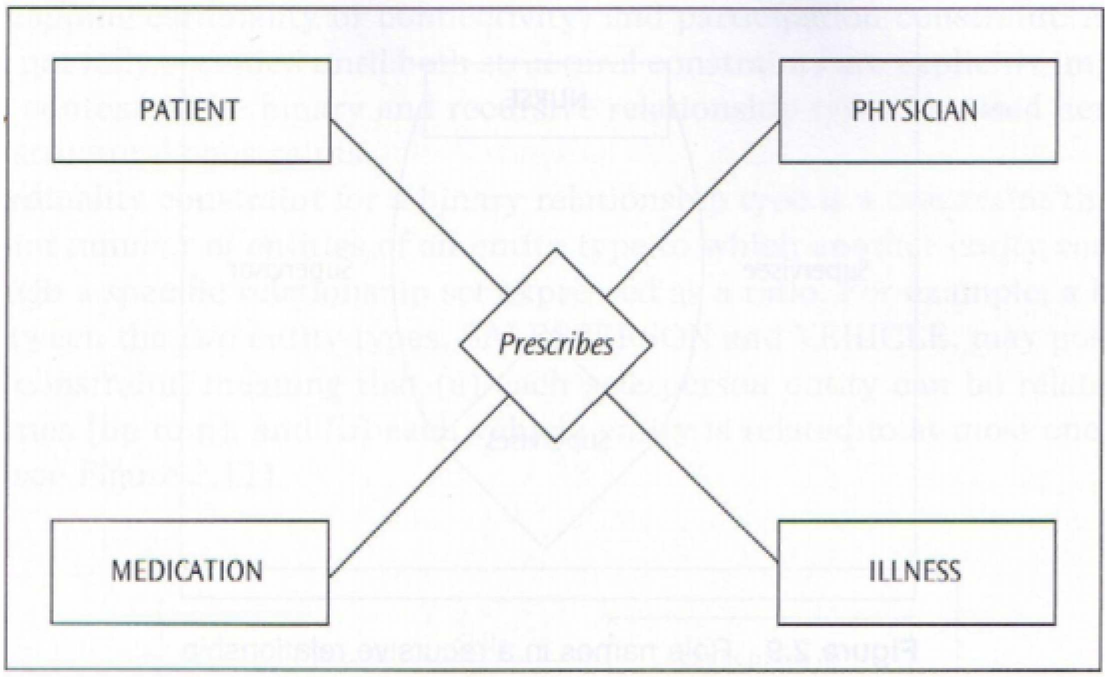


Figure 2.7 A quaternary relationship

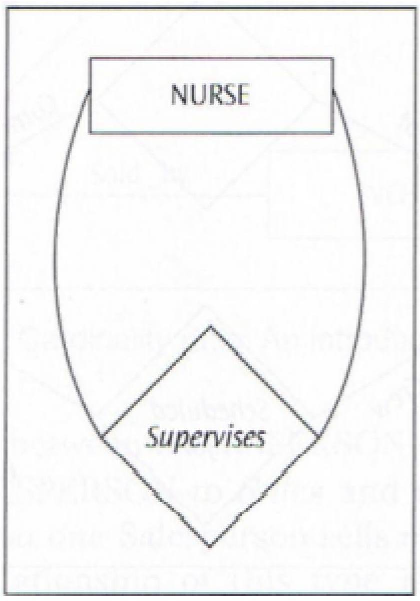


Figure 2.8 A recursive relationship