(Entity-Relationship Model)

Entity
*An **entity** is a "thing" or "object" in the real world that is distinguishable from all other objects.* It is usually a person, place, thing, or event to be recorded in the database. If the data model were a language, entities would be nouns. A university database may contain the following entities: *student, teacher, course, grade, and course_offered.* A point of sale (POS) database may contain the following entities: *employee, customer, product, supplier, and location.*
An entity has a set of properties, and the values for some set of properties may uniquely identify an entity. For instance, a person may have a *person-id* property whose value uniquely identifies that person.

Attributes
The properties of entities are called attributes. For example, a Branch entity may be described by the branch number (Branch_ID), address (Address), phone number (Tel_no), and fax number (Fax_no).

Simple and composite attributes
In our examples thus far, the attributes have been simple; that is, they are not divided into subparts. **Composite** attributes, on the other hand, can be divided into subparts (that is, other attributes). For example, an attribute *name* could be structured as a composite attribute consisting of *first-name*, *middle-initial*, and *last-name*. Using composite attributes in a design schema is a good choice if a user will wish to refer to an entire attribute on some occasions, and to only a component of the attribute on other occasions. Suppose we were to substitute for the *customer* entity-set attributes *customer-street* and *customer-city* the composite attribute *address* with the attributes *street*, *city*, *state*, and *zip-code*. Composite attributes help us to group together related attributes, making the modeling cleaner.
Note also that a composite attribute may appear as a hierarchy. In the composite attribute *address*, its component attribute *street* can be further divided into *street-number*, *street-name*, and *apartment-number*.

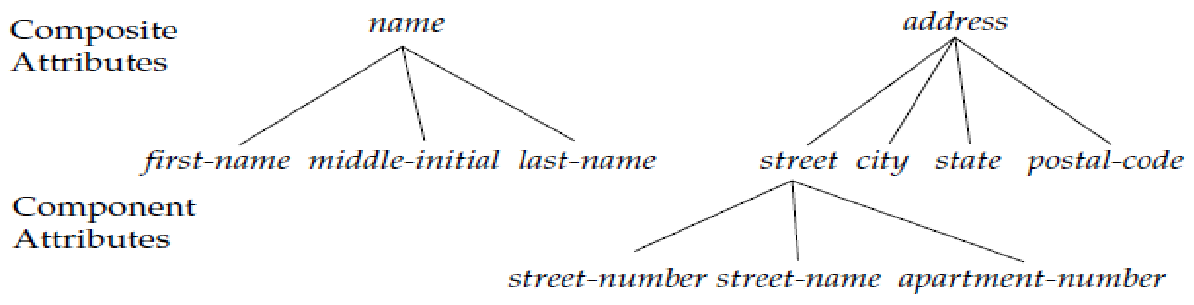


Figure 2.2 Composite attributes *customer-name* and *customer-address*.

Single-valued and multivalued attributes.

Single valued attributes: attributes with a single value for a particular entity are called single valued attributes.
Multivalued attributes : Attributes with a set of value for a particular entity are called multivalued attributes.
For instance, the *loan-number* attribute for a specific loan entity refers to only one loan number. Such attributes are said to be **single valued**. There may be instances where an attribute has a set of values for a specific entity. Consider an *employee* entity set with the attribute *phone-number*. An employee may have zero, one, or several phone numbers, and different employees may have different numbers of phones. This type of attribute is said to be **multivalued**.

Derived attribute.
The value for this type of attribute can be derived from the values of other related attributes or entities. For instance, let us say that the *customer* entity set has an attribute *loans-held*, which represents how many loans a customer has from the bank. We can derive the value for this attribute by counting the number of *loan* entities associated with that customer.

Null Value
An attribute takes a **null** value when an entity does not have a value for it. The *null* value may indicate "not applicable"—that is, that the value does not exist for the entity. For example, one may have no middle name. *Null* can also designate that an attribute value is unknown. An unknown value may be either *missing* (the value does exist, but we do not have that information) or *not known* (we do not know whether or not the value actually exists).
For instance, if the *name* value for a particular customer is *null*, we assume that the value is missing, since every customer must have a name. A null value for the *apartment-number* attribute could mean that the address does not include an apartment number (not applicable), that an apartment number exists but we do not know what it is (missing), or that we do not know whether or not an apartment number is part of the customer's address (unknown).

Week & Strong Entity
The entity set which does not have sufficient attributes to form a primary key is called as Weak entity set. An entity set that has a primary key is called as Strong entity set. Consider an entity set Payment which has three attributes: payment_number, payment_date and payment_amount. Although each payment entity is distinct but payment for different loans may share the same payment number. Thus, this entity set does not have a primary key and it is an entity set. Each weak set must be a part of one-to-many relationship set.

A member of a strong entity set is called dominant entity and member of weak entity set is called as subordinate entity. A weak entity set does not have a primary key but we need a means of distinguishing among all those entries in the entity set that depend on one particular strong entity set. The discriminator of a weak entity set is a set of attributes that allows this distinction be made. For example, payment_number acts as discriminator for payment entity set. It is also called as the Partial key of the entity set.

Relationship

A **relationship** is an association among several entities. For example, we can define a relationship that associates customer Hayes with loan L-15. This relationship specifies that Hayes is a customer with loan number L-15.

Mapping Cardinalities

Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set. Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets. In this section, we shall concentrate on only binary relationship sets. For a binary relationship set *R* between entity sets *A* and *B*, the mapping cardinality must be one of the following:

• **One to one**. An entity in *A* is associated with *at most* one entity in *B*, and an entity in *B* is associated with *at most* one entity in *A*. (See Figure 2.4a.)

• **One to many**. An entity in *A* is associated with any number (zero or more) of entities in *B*. An entity in *B*, however, can be associated with *at most* one entity in *A*. (See Figure 2.4b.)

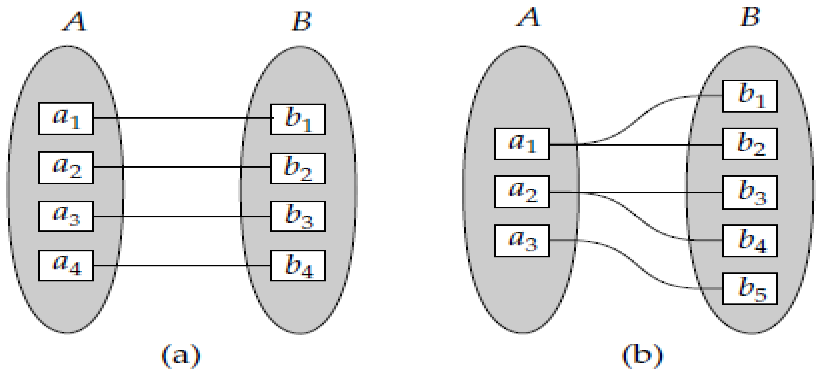


Figure 2.4 Mapping cardinalities. (a) One to one. (b) One to many.

• **Many to one**. An entity in *A* is associated with *at most* one entity in *B*. An entity in *B*, however, can be associated with any number (zero or more) of entities in *A*. (See Figure 2.5a.)

• **Many to many**. An entity in *A* is associated with any number (zero or more) of entities in *B*, and an entity in *B* is associated with any number (zero or more) of entities in *A*. (See Figure 2.5b.)

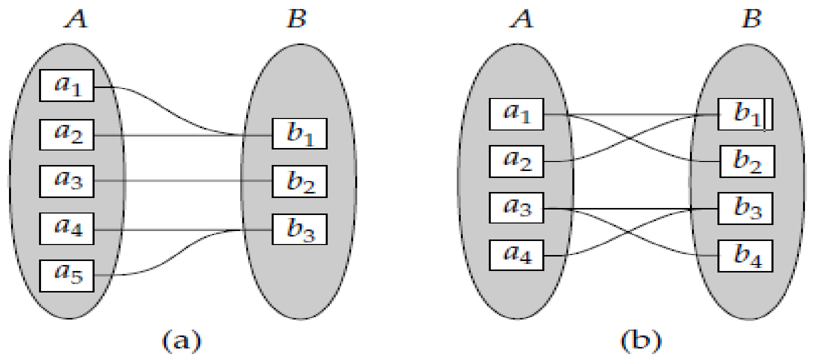


Figure 2.5 Mapping cardinalities. (a) Many to one. (b) Many to many.

Participation Constraints

The participation of an entity set E in a relationship set R is said to be total if everyentity in E participates in at least one relationship in R. If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be partial. For example, we expect every loan entity to be related to at least one customer through the borrower relationship. Therefore the participation of loan in the relationship set borrower is total. In contrast, an individual can be a bank customer whether or not she has a loan with the bank. Hence, it is possible that only some of the customer entities are related to the loan entity set through the borrower relationship, and the participation of customer in the borrower relationship set is therefore partial.

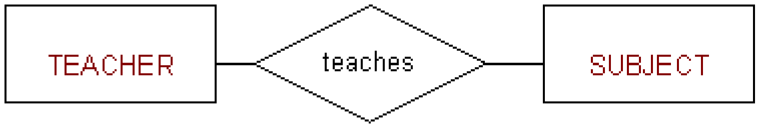
Binary versus n-ary Relationship Sets

Relationships in databases are often binary. Some relationships that appear to be nonbinary could actually be better represented by several binary relationships. For instance, one could create a ternary relationship parent, relating a child to his/her mother and father. However, such a relationship could also be represented by two binary relationships, mother and father, relating a child to his/her mother and father separately. Using the two relationships mother and father allows us record a child's mother, even if we are not aware of the father 's identity; a null value would be required if the ternary relationship parent is used. Using binary relationship sets is preferable in this case.

The degree of a relationship is the number of entity types that participate in the relationship. The three most common relationships in ER models are Binary, Unary and Ternary

A **binary relationship** is when two entities participate, and is the most common relationship degree.

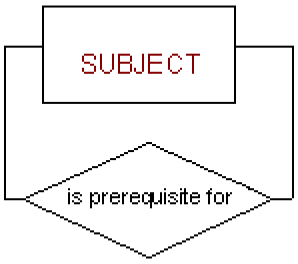
For Example:



A **unary relationship** is when both participants in the relationship are the same entity.

For Example:

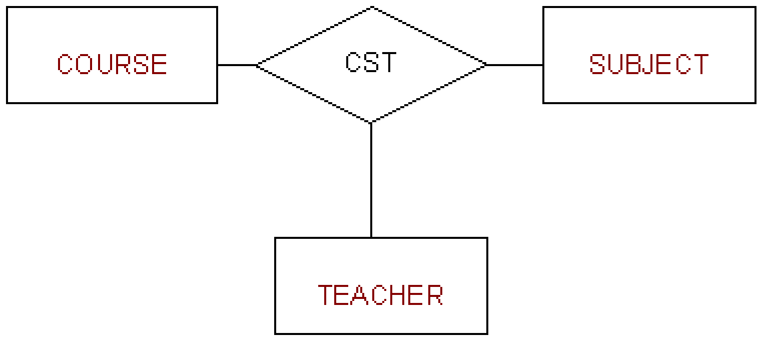
Subjects may be prerequisites for other subjects.



A **ternary relationship** is when three entities participate in the relationship.

For Example: **The University might need to record which teachers taught**

which subjects in which courses.



Entity-Relationship Diagram

an E-R diagram can express the overall logical structure of a database graphically. E-R diagrams are simple and clear — qualities that may well account in large part for the widespread use of the E-R model. Such a diagram consists of the following major components:

• Rectangles, which represent entity sets
• Ellipses, which represent attributes
• Diamonds, which represent relationship sets
• Lines, which link attributes to entity sets and entity sets to relationship sets
• Double ellipses, which represent multivalued attributes
• Dashed ellipses, which denote derived attributes
• Double lines, which indicate total participation of an entity in a relationship set
• Double rectangles, which represent weak entity sets

Extended E-R Features

Although the basic E-R concepts can model most database features, some aspects of a database may be more aptly expressed by certain extensions to the basic E-R model.

Specialization

An entity set may include subgroupings of entities that are distinct in some way from other entities in the set. The process of designating subgroupings within an entity set is called **specialization** For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set. The E-R model provides a means for representing these distinctive entity groupings. Consider an entity set *person*, with attributes *name*, *street*, and *city*. A person may be further classified as one of the following:

- ✓ *Customer*
- ✓ *employee*

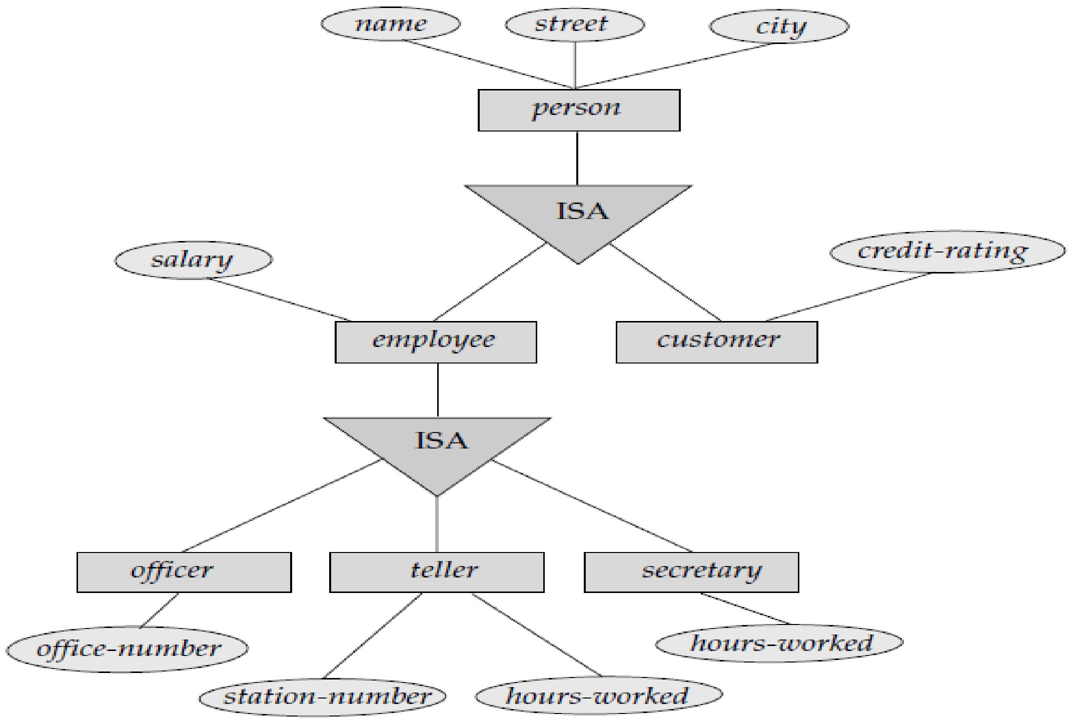


Figure 2.17 Specialization and generalization.

Generalization
Generalization is a simple inversion of specialization. There are similarities between the *customer* entity set and the *employee* entity set in the sense that they have several attributes in common. This commonality can be expressed by **generalization**, which is a containment relationship that exists between a *higher-level* entity set and one or more *lower-level* entity sets. In our example, *person* is the higher-level entity set and *customer* and *employee* are lower-level entity sets. Higher- and lower-level entity sets also may be designated by the terms **superclass** and **subclass**, respectively. The *person* entity set is the superclass of the *customer* and *employee* subclasses.

Constraints on Generalizations
To model an enterprise more accurately, the database designer may choose to place certain constraints on a particular generalization. One type of constraint involves determining which entities can be members of a given lower-level entity set. Such membership may be one of the following:

• **Condition-defined**. In condition-defined lower-level entity sets, membership is evaluated on the basis of whether or not an entity statistics an explicit condition or predicate. For example, assume that the higher-level entity set account has the attribute account-type. All account entities are evaluated on the defining account-type attribute. Only those entities that satisfy the condition account-type = “savings account” are allowed to belong to the lower-level entity set person. All entities that satisfy the condition account-type = “checking account” are included in checking account. Since all the lower-level entities are evaluated on the basis of the same attribute (in this case, on account-type), this type of generalization is said to be attribute-defined.

• **User-defined**. User-defined lower-level entity sets are not constrained by a membership condition; rather, the database user assigns entities to a given entity set. For instance, let us assume that, after 3 months of employment, bank employees are assigned to one of four work teams. We therefore represent the teams as four lower-level entity sets of the higher-level employee entity set. A given employee is not assigned to a specific team entity automatically on the basis of an explicit defining condition. Instead, the user in charge of this decision makes the team assignment on an individual basis. The assignment is implemented by an operation that adds an entity to an entity set.

A second type of constraint relates to whether or not entities may belong to more than one lower-level entity set within a single generalization. The lower-level entity sets may be one of the following:

• **Disjoint**. A disjointness constraint requires that an entity belong to no more than one lower-level entity set. In our example, an account entity can satisfy only one condition for the account-type attribute; an entity can be either a savings account or a checking account, but cannot be both.

• **Overlapping**. In overlapping generalizations, the same entity may belong to more than one lower-level entity set within a single generalization. For an illustration, consider the employee work team example, and assume that certain managers participate in more than one work team. A given employee may therefore appear in more than one of the team entity sets that are lower-level entity sets of employee. Thus, the generalization is overlapping.

As another example, suppose generalization applied to entity sets customer and employee leads to a higher-level entity set person. The generalization is overlapping if an employee can also be a customer. Lower-level entity overlap is the default case; a disjointness constraint must be placed explicitly on a generalization (or specialization). We can note a disjointedness constraint in an E-R diagram by adding the word disjoint next to the triangle symbol.

A final constraint, the completeness constraint on a generalization or specialization, specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization. This constraint may be one of the following:

• Total generalization or specialization. Each higher-level entity must belong to a lower-level entity set.• Partial generalization or specialization. Some higher-level entities may not belong to any lower-level entity set.

1. An employee may work on any number of projects (one, many, or none at all). While an employee is not required to be working on a project, each project must have at least one employee allocated to it (one or many). Furthermore, every employee must belong to one and only one department. For a department to exist however, it must have at least two employees. The data held on an employee is the first and last name, address, and the date

of joining. Each employee is given an employee number, which is unique throughout all branches of the company.

2. A small Cinema club runs a number of cinemas throughout the region. Each cinema can show various films at several daily showings over several days/weeks/months depending on the popularity of the film. Members book tickets for a particular showing.

3. Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.

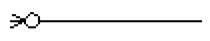
4. Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.

Crows Foot Notation

Cardinality and Participation Constraint are the indicators of the business rules around a relationship. Cardinality refers to the maximum number of times an instance in one entity can be associated with instances in the related entity. Participation Constraint refers to the minimum number of times an instance in one entity can be associated with an instance in the related entity.

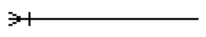
Cardinality can be 1 or Many and the symbol is placed on the outside ends of the relationship line, closest to the entity, Participation Constraints can be 1 or 0 and the symbol is placed on the inside, next to the cardinality symbol. For a cardinality of 1 a straight line is drawn. For a cardinality of Many a foot with three toes is drawn. For a Participation Constraint of 1 a straight line is drawn. For a modality of 0 a circle is drawn.

The crow's foot notation was invented by Gordon Everest, who originally used the term "inverted arrow" but now just calls it a "fork". For cardinality, a fork or crow's foot intuitively indicates "many", by its many "toes".



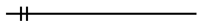
zero or more

Cardinality is many, Participation Constraint or optionality is zero



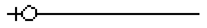
1 or more

cardinality is many, Participation Constraint or optionality is 1



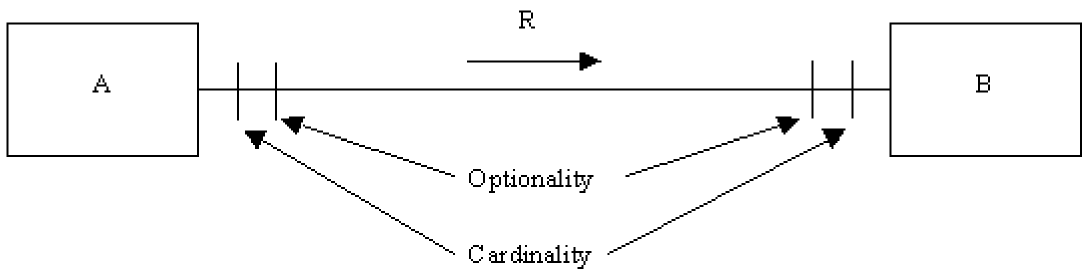
1 and only 1 (exactly 1)

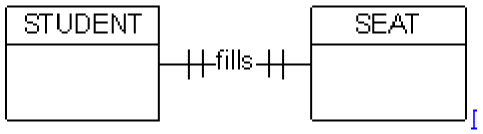
cardinality is 1, Participation Constraint or optionality is 1



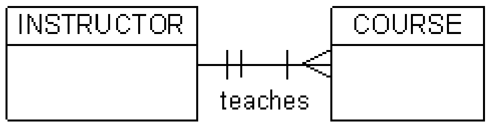
zero or 1

cardinality is 1, Participation Constraint or optionality is zero

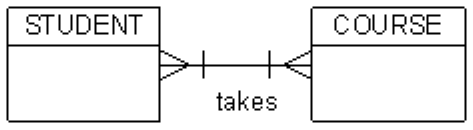




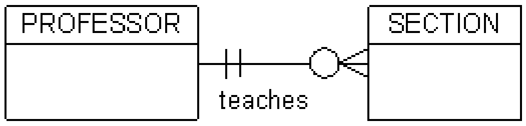
A student fills a seat. This is a 1:1 relationship. 1 student can fill a maximum of 1 seat. 1 seat can be filled by a maximum of 1 student. Each side of the relationship has a cardinality of 1. The participation constraint on each side is also 1. A student must fill at least 1 seat, and 1 seat must be filled by at least one student. Although it may seem possible to have an empty seat, in which case the participation constraint would be 0, the business rules we have defined determine that for the purposes of our database. We do not want to be able to store information about empty seats.



An instructor teaches a course. This is a 1:M relationship. One instructor can teach many courses, but one course can only be taught by one instructor. The participation constraint is one on both ends of the relationship. According to the business rules we have defined. An instructor must teach at least 1 course, and a course must be taught by 1 instructor.

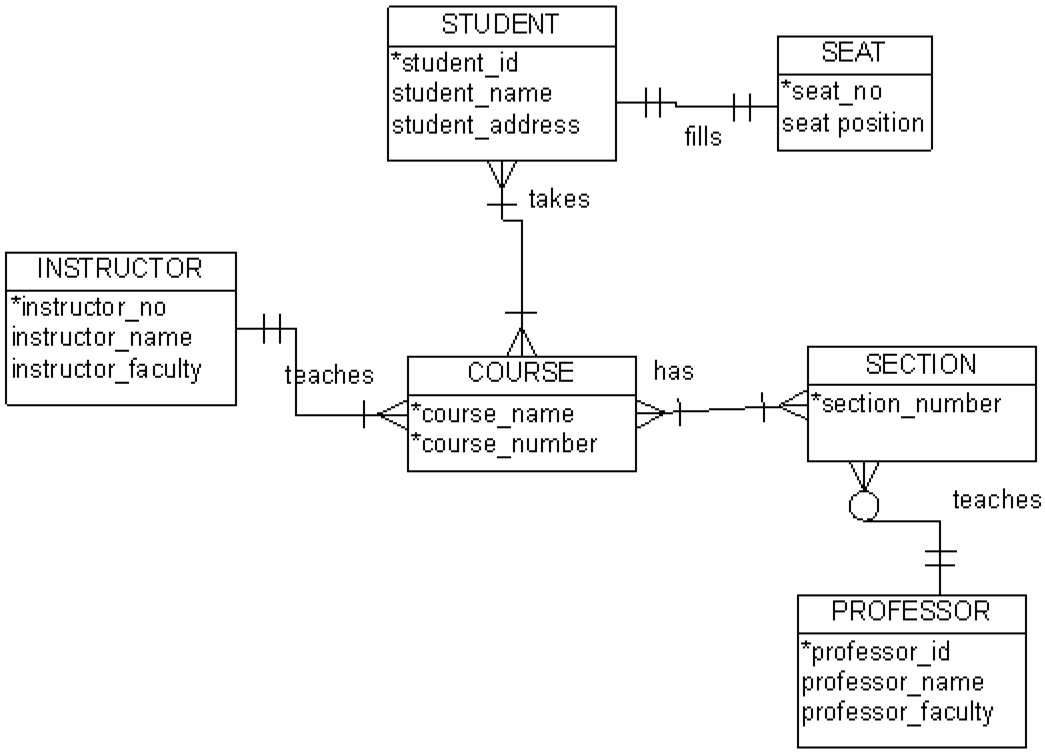


A student takes a course. This is a M:M relationship. One student can take many courses, and one course can be taken by many students. For the purposes of the information we want to store, the business rules indicate that a student must take at least one course, and a course must be taken by at least one student. The participation constraint is 1 on both sides.

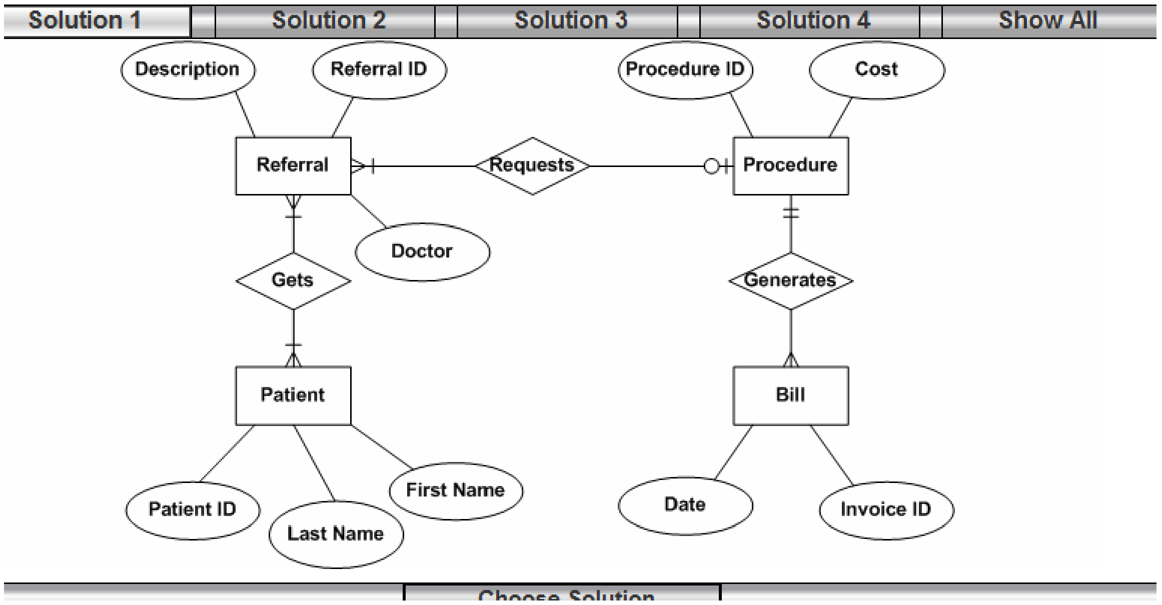


A professor teaches a section of a course. This is a 1:M relationship. One professor can teach many sections of a course, and 1 section of a course can only be taught by 1 instructor. On the participation constraint side of things a professor does not need to teach a section at all, so the participation constraint is zero on that side of the relationship. On the other hand, a section must be taught by 1 instructor.

Typically, ERDs are much more complex than this, involving quite a number of entities and relationships. If we join all of the above relationships together and add a few attributes, a small collection of data might be depicted in the following way using Crows Foot Notation:



You have been hired by a regional hospital to create a database that will track the procedures performed for patients and the corresponding billing information. To ease the storing of this information, the hospital assigns each new patient a unique patient number. Often, the same patient may be seen by the hospital for different procedures, and will, therefore, keep the same unique number. The hospital needs a database system that is able to track all procedures performed for a patient and can bill him or her individually for the visits.


Solution 1
Solution 2
Solution 3
Solution 4
Show All
Description
Referral ID
Procedure ID
Cost
Referral
Requests
Procedure
Doctor
Gets
Generates
Patient
Bill
Patient ID
Last Name
First Name
Date
Invoice ID
Choose Solution