7.1 Overview of the Design Process

The task of creating a database application is a complex one, involving design of the database schema, design of the programs that access and update the data, and design of a security scheme to control access to data.

7.1.1 Design Phases

For small applications, it may be feasible for a database designer who understands the application requirements to decide directly on the relations to be created, their attributes, and constraints on the relations. However, such a direct design process is difficult for real-world applications, since they are often highly complex. Often no one person understands the complete data needs of an application. The database designer must interact with users of the application to understand the needs of the application,

- The initial phase of database design is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements.
- Next, the designer chooses a data model and, by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database. The schema developed at this **conceptual-design** phase provides a detailed overview of the enterprise. The entity-relationship model, which we study in the rest of this chapter, is typically used to represent the conceptual design. Typically, the conceptual-design phase re- sults in the creation of an entity-relationship diagram that provides a graphic representation of the schema.

  The designer reviews the schema to confirm that all data requirements are indeed satisfied and are not in conflict with one another. She can also examine the design to remove any redundant features. Her focus at this point is on describing the data and their relationships, rather than on specifying physical storage details.
- A fully developed conceptual schema also indicates the functional requirements of the enterprise. In a **specification of functional requirements**, users describe the kinds of operations (or transactions) that will be performed on the data. Example operations include modifying or updating data, searching for and retrieving specific data, and deleting data. At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements.
- The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases.
  - In the **logical-design phase**, the designer maps the high-level conceptual

schema onto the implementation data model of the database system that

will be used. The implementation data model is typically the relational data model, and this step typically consists of mapping the conceptual schema defined using the entity-relationship model into a relation schema.

- Finally, the designer uses the resulting system-specific database schema in the subsequent **physical-design phase**, in which the physical features of the database are specified. These features include the form of file or- ganization and choice of index structures.

7.1.2 Design Alternatives

A major part of the database design process is deciding how to represent in the design the various types of "things" such as people, places, products, and the like. We use the term *entity* to refer to any such distinctly identifiable item.

In designing a database schema, we must ensure that we avoid two major pitfalls:

1. **Redundancy:** A bad design may repeat information. For example, if we store the course identifier and title of a course with each course offering, the title would be stored redundantly (that is, multiple times, unnecessarily) with each course offering. It would suffice to store only the course identifier with each course offering, and to associate the title with the course identifier only once, in a course entity.

   The biggest problem with such redundant representation of information is that the copies of a piece of information can become inconsistent if the information is updated without taking precautions to update all copies of the information.

2. **Incompleteness:** A bad design may make certain aspects of the enterprise difficult or impossible to model. For example, suppose that, as in case (1) above, we only had entities corresponding to course offering, without hav- ing an entity corresponding to courses. Equivalently, in terms of relations, suppose we have a single relation where we repeat all of the course infor- mation once for each section that the course is offered. It would then be impossible to represent information about a new course, unless that course is offered. We might try to make do with the problematic design by stor- ing null values for the section information. Such a work-around is not only unattractive, but may be prevented by primary-key constraints.

7.2 The Entity-Relationship Model

The **entity-relationship** (E-R) data model was developed to facilitate database design by allowing specification of an *enterprise schema* that represents the overall logical structure of a database.

The E-R data model employs three basic concepts:

- entity sets,
- relationship sets, and
- attributes.

The E-R model also has an associated diagrammatic representation, the E-R diagram.

7.2.1 Entity Sets

An entity is a "thing" or "object" in the real world that is distinguishable from all other objects. For example, each person in a university is an entity. 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. Thus, the value 677-89-9011 for *person id* would uniquely identify one particular person in the university. An entity may be concrete, such as a person or a book, or it may be abstract, such as a course, a course offering, or a flight reservation.

An entity set is a set of entities of the same type that share the same properties, or attributes. The set of all people who are instructors at a given university, for example, can be defined as the entity set *instructor*.

An entity is represented by a set of **attributes**. **Attributes are descriptive properties possessed by each member of an entity set.** Possible attributes of the *instructor* entity set are *ID*, *name*, *dept name*, and *salary*.

A database thus includes a collection of entity sets, each of which contains any number of entities of the same type. Figure 7.1 shows part of a university database that consists of two entity sets: *instructor* and *student*.

76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Figure 7.1 Entity sets *instructor* and *student*.

with attributes *course_id*, *title*, *dept_name* and *credits*. In a real setting, a university

database may keep dozens of entity sets.

7.2.2 Relationship Sets

A relationship is an association among several entities. For example, we can define a relationship *advisor* that associates instructor Katz with student Shankar. This relationship specifies that Katz is an advisor to student Shankar.

A **relationship set** is a set of relationships of the same type. Formally, it is a mathematical relation on $n \geq 2$ (possibly nondistinct) entity sets. If $E_1, E_2, \ldots, E_n$ are entity sets, then a relationship set R is a subset of

$$\{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

where $(e_1, e_2, \ldots, e_n)$ is a relationship.

Consider the two entity sets *instructor* and *student* in Figure 7.1. We define the relationship set *advisor* to denote the association between instructors and students. Figure 7.2 depicts this association.

The association between entity sets is referred to as participation; that is, the entity sets $E_1, E_2, \ldots, E_n$ **participate** in relationship set R. A **relationship instance** in an E-R schema represents an association between the named entities in the real-world enterprise that is being modeled.

The function that an entity plays in a relationship is called that entity's **role**. Since entity sets participating in a relationship set are generally distinct, roles

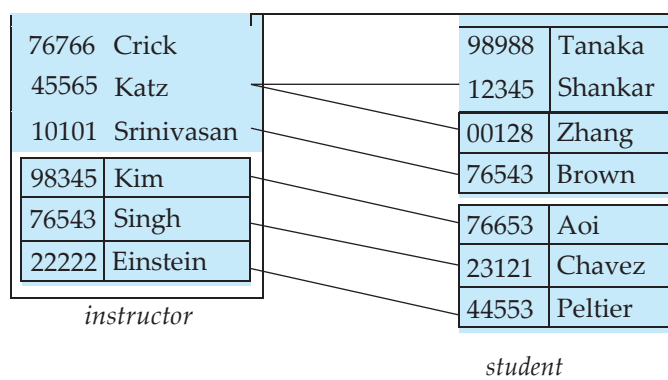
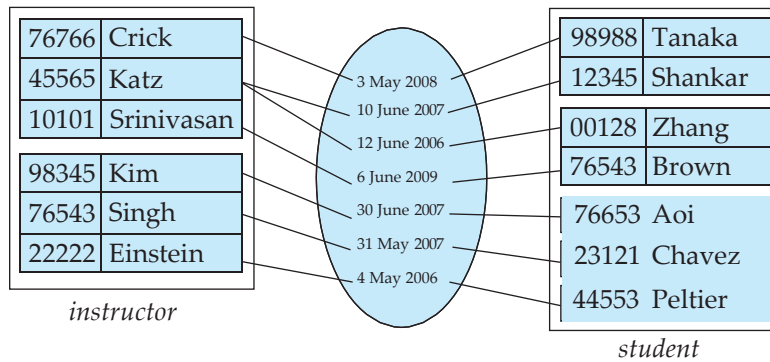


Figure 7.2 Relationship set *advisor*.

are implicit and are not usually specified. However, they are useful when the meaning of a relationship needs clarification. Such is the case when the entity sets of a relationship set are not distinct; that is, the same entity set participates in a relationship set more than once, in different roles. In this type of relationship set, sometimes called a **recursive** relationship set, explicit role names are necessary to specify how an entity participates in a relationship instance.

A relationship instance in a given relationship set must be uniquely identifiable from its participating entities, without using the descriptive attributes. To understand this point, suppose we want to model all the dates when an instructor


76766 Crick
45565 Katz
10101 Srinivasan
98345 Kim
76543 Singh
22222 Einstein
instructor
3 May 2008
10 June 2007
12 June 2006
6 June 2009
30 June 2007
31 May 2007
4 May 2006
98988 Tanaka
12345 Shankar
00128 Zhang
76543 Brown
76653 Aoi
23121 Chavez
44553 Peltier
student

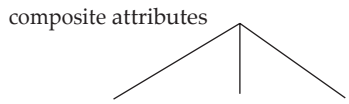
became an advisor of a particular student. **The single-valued attribute *date* can store a single *date* only.**

7.2.3 Attributes

For each attribute, **there is a set of permitted values, called the domain, or value set, of that attribute.** The domain of attribute *course id* might be the set of all text strings of a certain length. Similarly, the domain of attribute *semester* might be strings from the set Fall, Winter, Spring, Summer .

Formally, **an attribute of an entity set is a function that maps from the entity set into a domain.**

- **Simple** and **composite** attributes. **The attributes have been simple; that is, they have not been 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 to add an address to the *student* entity-set. The address can be defined as the composite attribute *address* with the attributes *street*, *city*, *state*, and *zip code*.[3]
- **Single-valued** and **multivalued** attributes. **The attributes in our examples all have a single value for a particular entity.** For instance, the *student ID* attribute for a specific student entity refers to only one student *ID*. Such attributes are said to be **single valued**. There may be instances where an attribute has a set of values for a specific entity. Suppose we add to the *instructor* entity set


composite attributes

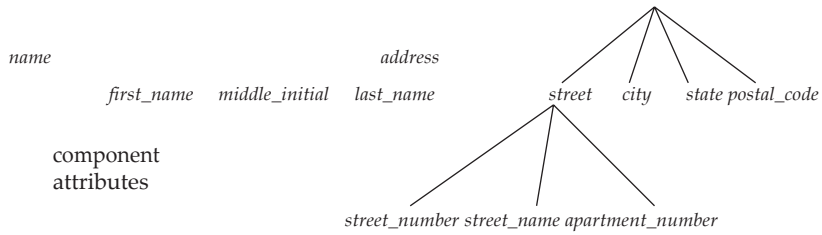


Figure 7.4 Composite attributes instructor *name* and *address*.

a *phone_number* attribute. An *instructor* may have zero, one, or several phone numbers, and different instructors 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.

  If the *instructor* entity set also has an attribute *date of birth*, we can calculate *age* from *date of birth* and the current date. Thus, *age* is a derived attribute. In this case, *date of birth* may be referred to as a *base* attribute, or a *stored* attribute.

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* or *not known.*

7.3 Constraints

An E-R enterprise schema may define certain constraints to which the contents of a database must conform.

7.3.1 Mapping Cardinalities

Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.

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 7.5a.)
- **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 7.5b.)

A *B* *A* *B*

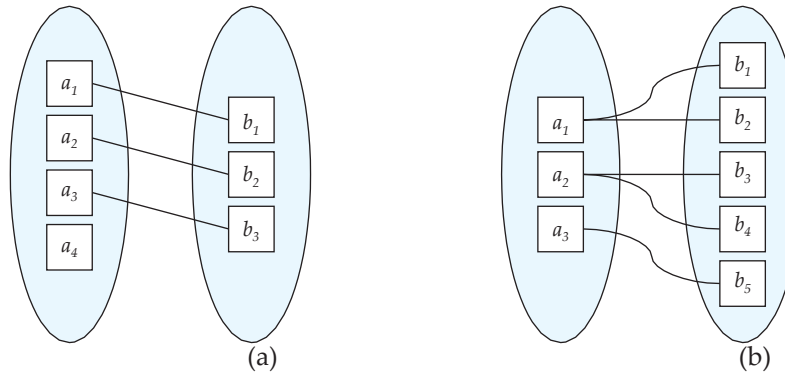


Figure 7.5 Mapping cardinalities. (a) One-to-one. (b) One-to-many.

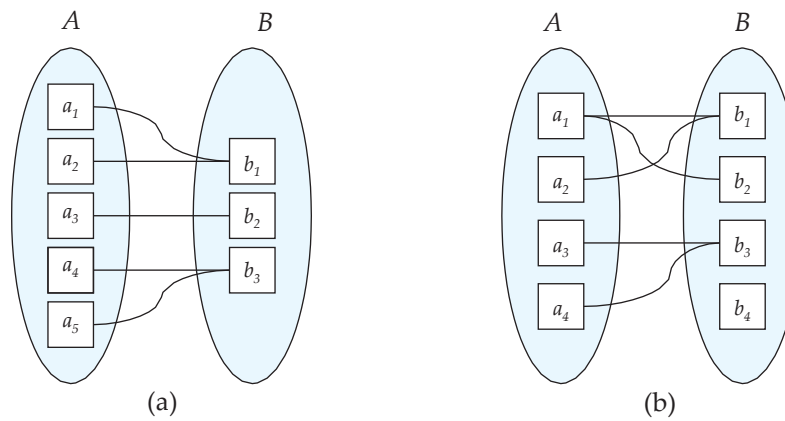


Figure 7.6 Mapping cardinalities. (a) Many-to-one. (b) Many-to-many.

- **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 7.6a.)
- **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 7.6b.)

7.3.2 Participation Constraints

The participation of an entity set E in a relationship set R is said to be **total** if every entity 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**. In Figure 7.5a, the participation of B in the relationship set is total while the participation of A in the relationship set is partial. In Figure 7.5b, the participation of both A and B in the relationship set are total.

7.3.3 Keys

Therefore, the values of the attribute values of an entity must be such that they can *uniquely identify* the entity. In other words, no two entities in an entity set are allowed to have exactly the same value for all attributes.

A key for an entity is a set of attributes that suffice to distinguish entities from each other. The concepts of superkey, candidate key, and primary key are applicable to entity sets just as they are applicable to relation schemas.

Keys also help to identify relationships uniquely, and thus distinguish relationships from each other.

The primary key of an entity set allows us to distinguish among the various entities of the set.

Let R be a relationship set involving entity sets $E_1, E_2, \ldots, E_n$. Let *primary-key*(E_i) denote the set of attributes that forms the primary key for entity set E_i. Assume for now that the attribute names of all primary keys are unique. The composition of the primary key for a relationship set depends on the set of attributes associated with the relationship set R.

If the relationship set R has no attributes associated with it, then the set of attributes

$$\textit{primary-key}(E_1) \cup \textit{primary-key}(E_2) \cup \cdots \cup \textit{primary-key}(E_n)$$

describes an individual relationship in set R.

If the relationship set R has attributes $a_1, a_2, \ldots, a_m$ associated with it, then the set of attributes

$$\textit{primary-key}(E_1) \cup \textit{primary-key}(E_2) \cup \cdots \cup \textit{primary-key}(E_n) \cup \{a_1, a_2, \ldots, a_m\}$$

describes an individual relationship in set R.

In both of the above cases, the set of attributes

$$\textit{primary-key}(E_1) \cup \textit{primary-key}(E_2) \cup \cdots \cup \textit{primary-key}(E_n)$$

forms a superkey for the relationship set.

If the attribute names of primary keys are not unique across entity sets, the attributes are renamed to distinguish them; the name of the entity set combined with the name of the attribute would form a unique name. If an entity set participates more than once in a relationship set (as in the *prereq* relationship in

Section 7.2.2), the role name is used instead of the name of the entity set, to form a unique attribute name.

The structure of the primary key for the relationship set depends on the mapping cardinality of the relationship set. As an illustration, consider the entity sets *instructor* and *student*, and the relationship set *advisor*, with attribute *date*, in Section 7.2.2. Suppose that the relationship set is many-to-many. Then the primary key of *advisor* consists of the union of the primary keys of *instructor* and *student*. If the relationship is many-to-one from *student* to *instructor* — that is, each student can have have at most one advisor — then the primary key of *advisor* is simply the primary key of *student*. However, if an instructor can advise only one student — that is, if the *advisor* relationship is many-to-one from *instructor* to *student* — then the primary key of *advisor* is simply the primary key of *instructor*. For one-to-one relationships either candidate key can be used as the primary key.

7.4 Removing Redundant Attributes in Entity Sets

When we design a database using the E-R model, we usually start by identifying those entity sets that should be included. Once the entity sets are decided upon, we must choose the appropriate attributes. These attributes are supposed to represent the various values we want to capture in the database.

Once the entities and their corresponding attributes are chosen, the relation- ship sets among the various entities are formed. To illustrate, consider the entity sets *instructor* and *department*:

- The entity set *instructor* includes the attributes *ID*, *name*, *dept name*, and *salary*, with *ID* forming the primary key.
- The entity set *department* includes the attributes *dept name*, *building*, and *bud- get*, with *dept name* forming the primary key.

We model the fact that each instructor has an associated department using a relationship set *inst dept* relating *instructor* and *department*.

The attribute *dept name* appears in both entity sets. Since it is the primary key for the entity set *department*, it is redundant in the entity set *instructor* and needs to be removed.

Removing the attribute *dept name* from the *instructor* entity set may appear rather unintuitive, since the relation *instructor* that we used in the earlier chap- ters had an attribute *dept name*. As we shall see later, when we create a relational schema from the E-R diagram, the attribute *dept name* in fact gets added to the relation *instructor*, but only if each instructor has at most one associated depart- ment. If an instructor has more than one associated department, the relationship between instructors and departments is recorded in a separate relation *inst dept*. Treating the connection between instructors and departments uniformly as a relationship, rather than as an attribute of *instructor*, makes the logical relationship explicit, and helps avoid a premature assumption

that each instructor is associated with only one department.

- The entity set *section* includes the attributes *course id*, *sec id*, *semester*, *year*, *building*, *room number*, and *time slot id*, with (*course id*, *sec id*, *year*, *semester*) forming the primary key.
- The entity set *time slot* includes the attributes *time slot id*, which is the primary key,[4] and a multivalued composite attribute {(*day*, *start time*, *end time*)}.[5]

These entities are related through the relationship set *sec time slot*.

The attribute *time slot id* appears in both entity sets. Since it is the primary key for the entity set *time slot*, it is redundant in the entity set *section* and needs to be removed.

As a final example, suppose we have an entity set *classroom*, with attributes *building*, *room number*, and *capacity*, with *building* and *room number* forming the primary key. Suppose also that we have a relationship set *sec class* that relates *section* to *classroom*. Then the attributes *building*, *room number* are redundant in the entity set *section*.

A good entity-relationship design does not contain redundant attributes. For our university example, we list the entity sets and their attributes below, with primary keys underlined

- **classroom**: with attributes (*building*, *room number*, *capacity*).
- **department**: with attributes (*dept name*, *building*, *budget*).
- **course**: with attributes (*course id*, *title*, *credits*).
- **instructor**: with attributes (*ID*, *name*, *salary*).
- **section:** with attributes (*course id*, *sec id*, *semester*, *year*).
- **student**: with attributes (*ID*, *name*, *tot cred*).
- **time slot**: with attributes (*time slot id*, {(*day*, *start time*, *end time*) }). The relationship sets in our design are listed below:

- **inst dept**: relating instructors with departments.
- **stud dept**: relating students with departments.
- **teaches**: relating instructors with sections.
- **takes**: relating students with sections, with a descriptive attribute *grade*.
- **course dept**: relating courses with departments.
- **sec course**: relating sections with courses.
- **sec class**: relating sections with classrooms.
- **sec time slot**: relating sections with time slots.

- **advisor**: relating students with instructors.
- **prereq**: relating courses with prerequisite courses.

7.4 Entity-Relationship Diagrams

As we saw briefly in Section 1.3.3, 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.

7.4.1 Basic Structure

An E-R diagram consists of the following major components:

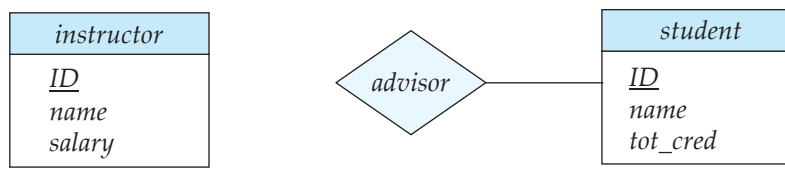


Figure 7.7 E-R diagram corresponding to instructors and students.

- **Rectangles divided into two parts** represent entity sets. The first part, which in this textbook is shaded blue, contains the name of the entity set. The second part contains the names of all the attributes of the entity set.
- **Diamonds** represent relationship sets.
- **Undivided rectangles** represent the attributes of a relationship set. Attributes that are part of the primary key are underlined.
- **Lines** link entity sets to relationship sets.
- **Dashed lines** link attributes of a relationship set to the relationship set.
- **Double lines** indicate total participation of an entity in a relationship set.
- **Double diamonds** represent identifying relationship sets linked to weak entity sets (we discuss identifying relationship sets and weak entity sets later, in Section 7.5.6).

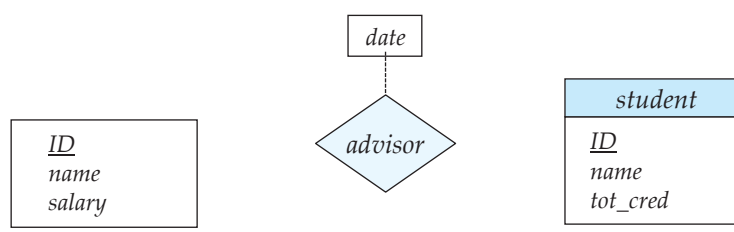


Figure 7.8 E-R diagram with an attribute attached to a relationship set.

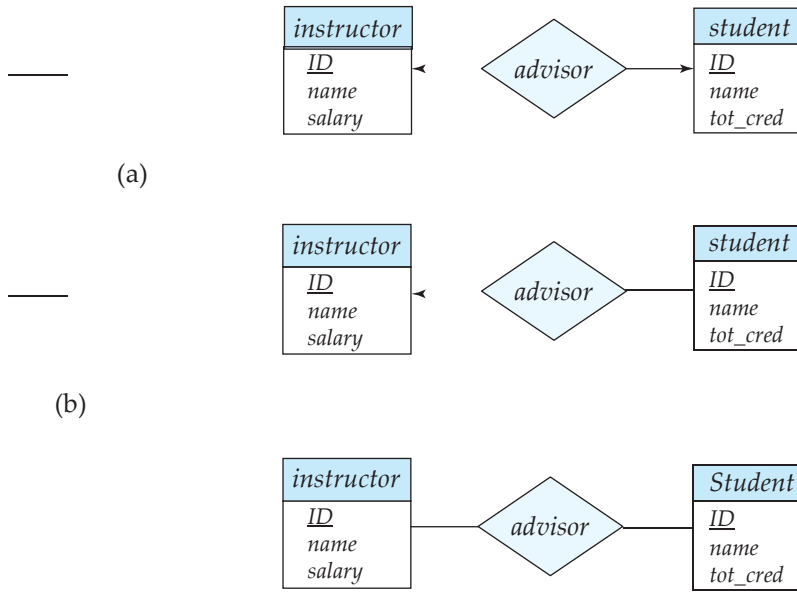


Figure 7.9 Relationships. (a) One-to-one. (b) One-to-many. (c) Many-to-many.

7.4.2 Mapping Cardinality

The relationship set *advisor*, between the *instructor* and *student* entity sets may be one-to-one, one-to-many, many-to-one, or many-to-many. To distinguish among these types, we draw either a directed line ($\rightarrow$) or an undirected line (—) between the relationship set and the entity set in question, as follows:

- **One-to-one:** We draw a directed line from the relationship set *advisor* to both entity sets *instructor* and *student* (see Figure 7.9a). This indicates that an instructor may advise at most one student, and a student may have at most one advisor.
- **One-to-many:** We draw a directed line from the relationship set *advisor* to the entity set *instructor* and an undirected line to the entity set *student* (see Figure 7.9b). This indicates that an instructor may advise many students, but a student may have at most one advisor.
- **Many-to-one:** We draw an undirected line from the relationship set *advisor* to the entity set *instructor* and a directed line to the entity set *student*. This indicates that an instructor may advise at most one student, but a student may have many advisors.
- **Many-to-many:** We draw an undirected line from the relationship set *advisor* to both entity sets *instructor* and *student* (see Figure 7.9c). This indicates that an instructor may advise many students, and a student may have many advisors.

7.4.3 Complex Attributes

Figure 7.11 shows how composite attributes can be represented in the E-R notation. Here, a composite attribute *name*, with component attributes *first_name*, *middle_initial*, and *last_name* replaces the simple attribute *name* of *instructor*. As another example, suppose we were to add an address to the *instructor* entity-set. The address can be defined as the composite attribute *address* with the attributes

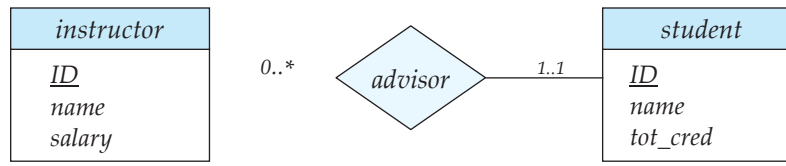


Figure 7.10 Cardinality limits on relationship sets.

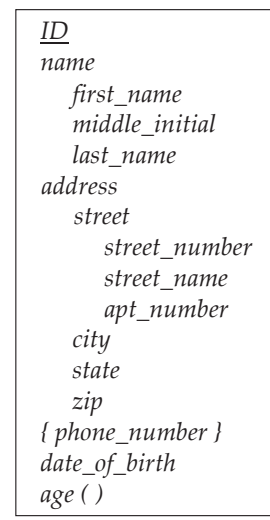


Figure 7.11 E-R diagram with composite, multivalued, and derived attributes.

street, *city*, *state*, and *zip code*. The attribute *street* is itself a composite attribute whose component attributes are *street number*, *street name*, and *apartment number*. Figure 7.11 also illustrates a multivalued attribute *phone number*, denoted by "*{phone_number}*", and a derived attribute *age*, depicted bya "*age* ( )".

7.4.4 Roles

We indicate roles in E-R diagrams by labeling the lines that connect diamonds to rectangles. Figure 7.12 shows the role indicators *course_id* and *prereq_id* between the *course* entity set and the *prereq* relationship set.

7.4.5 Nonbinary Relationship Sets

Nonbinary relationship sets can be specified easily in an E-R diagram. Figure 7.13 consists of the three entity sets *instructor*, *student*, and *project*, related through the relationship set *proj_guide*.

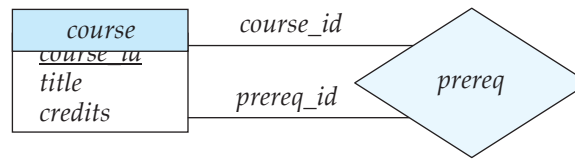


Figure 7.12 E-R diagram with role indicators.

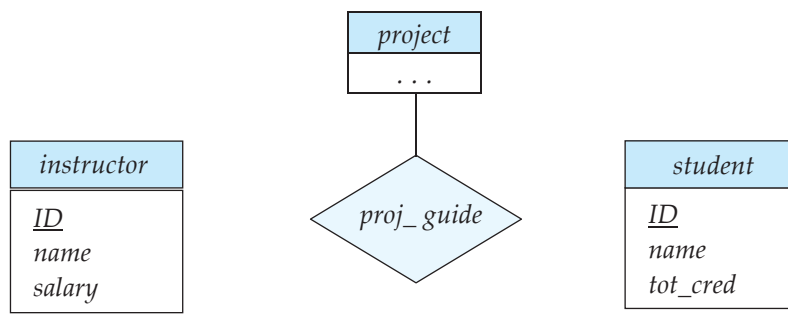


Figure 7.13 E-R diagram with a ternary relationship.

We can specify some types of many-to-one relationships in the case of nonbinary relationship sets. Suppose a *student* can have at most one instructor as a guide on a project. This constraint can be specified by an arrow pointing to *instructor* on the edge from *proj_guide*.

We permit at most one arrow out of a relationship set, since an E-R diagram with two or more arrows out of a nonbinary relationship set can be interpreted in two ways. Suppose there is a relationship set R between entity sets $A_1, A_2, \ldots, A_n$, and the only arrows are on the edges to entity sets $A_{i+1}, A_{i+2}, \ldots, A_n$. Then, the two possible interpretations are:

1. A particular combination of entities from $A_1, A_2, \ldots, A_i$ can be associated with at most one combination of entities from $A_{i+1}, A_{i+2}, \ldots, A_n$. Thus, the primary key for the relationship R can be constructed by the union of the primary keys of $A_1, A_2, \ldots, A_i$.
2. For each entity set A_k, $i < k \leq n$, each combination of the entities from the other entity sets can be associated with at most one entity from A_k. Each set $\{A_1, A_2, \ldots, A_{k-1}, A_{k+1}, \ldots, A_n\}$, for $i < k \leq n$, then forms a candidate key.

7.4.6 Weak Entity Sets

Consider a *section* entity, which is uniquely identified by a course identifier, semester, year, and section identifier. Clearly, section entities are related to course

entities. Suppose we create a relationship set *sec course* between entity sets *section* and *course*.

Now, observe that the information in *sec course* is redundant, since *section* already has an attribute *course id*, which identifies the course with which the section is related. One option to deal with this redundancy is to get rid of the relationship *sec course*; however, by doing so the relationship between *section* and *course* becomes implicit in an attribute, which is not desirable.

The notion of *weak entity set* formalizes the above intuition. An entity set that does not have sufficient attributes to form a primary key is termed a **weak entity set**. An entity set that has a primary key is termed a **strong entity set**.

For a weak entity set to be meaningful, it must be associated with another entity set, called the **identifying** or **owner entity set**. Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set. The identifying entity set is said to **own** the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**.

The identifying relationship is many-to-one from the weak entity set to the identifying entity set, and the participation of the weak entity set in the rela- tionship is total. The identifying relationship set should not have any descriptive attributes, since any such attributes can instead be associated with the weak entity set.

Although a weak entity set does not have a primary key, we nevertheless need a means of distinguishing among all those entities in the weak entity set that depend on one particular strong entity.

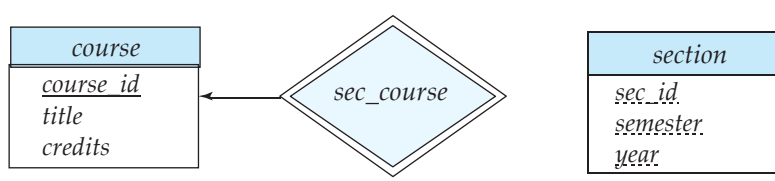


Figure 7.14 E-R diagram with a weak entity set.

had a primary key. However, conceptually, a *section* is still dependent on a *course* for its existence, which is made explicit by making it a weak entity set.

In E-R diagrams, a weak entity set is depicted via a rectangle, like a strong entity set, but there are two main differences:

- The discriminator of a weak entity is underlined with a dashed, rather than a solid, line.
- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond.

In Figure 7.14, the weak entity set *section* depends on the strong entity set *course*

via the relationship set *sec_course*.

The figure also illustrates the use of double lines to indicate *total participation*; the participation of the (weak) entity set *section* in the relationship *sec course* is total, meaning that every section must be related via *sec course* to some course. Finally, the arrow from *sec_course* to *course* indicates that each section is related to a single course.

A weak entity set can participate in relationships other than the identifying relationship. A weak entity set may participate as owner in an identifying relationship with another weak entity set. **The primary key of the weak entity set would consist of the union of the primary keys of the identifying entity sets, plus the discriminator of the weak entity set.**

7.4.7 E-R diagram for the University Enterprise

In Figure 7.15, we show an E-R diagram that corresponds to the university enterprise that we have been using thus far in the text.

In our university database, we have a constraint that each instructor must have exactly one associated department. As a result, there is a double line in Figure 7.15 between *instructor* and *inst_dept*, indicating total participation of *instructor* in *inst dept*; that is, each instructor must be associated with a department. Further, there is an arrow from *inst_dept* to *department*, indicating that each instructor can have at most one associated department.

7.5 Reduction to Relational Schemas

For each entity set and for each relationship set in the database design, there is a unique relation schema to which we assign the name of the corresponding entity set or relationship set.

7.5.1 Representation of Strong Entity Sets with Simple Attributes

Let E be a strong entity set with only simple descriptive attributes a_1, $a_2, \ldots, a_n$. We represent this entity by a schema called E with n distinct attributes. Each tuple in a relation on this schema corresponds to one entity of the entity set E.

For schemas derived from strong entity sets, the primary key of the entity set serves as the primary key of the resulting schema. This follows directly from the fact that each tuple corresponds to a specific entity in the entity set.

As an illustration, consider the entity set *student* of the E-R diagram in Fig- ure 7.15. This entity set has three attributes: *ID*, *name*, *tot cred*. We represent this entity set by a schema called *student* with three attributes:

student (*ID*, *name*, *tot cred*)

Note that since student *ID* is the primary key of the entity set, it is also the primary key of the relation schema.

Continuing with our example, for the E-R diagram in Figure 7.15, all the strong entity sets, except *time slot*, have only simple attributes. The schemas derived from these strong entity sets are:

classroom (*building*, *room_number*, *capacity*)

department (*dept name*, *building*, *budget*)
course (*course id*, *title*, *credits*)
instructor (*ID*, *name*, *salary*)
student (*ID*, *name*, *tot_cred*)

As you can see, both the *instructor* and *student* schemas are different from the schemas we have used in the previous chapters (they do not contain the attribute *dept_name*). We shall revisit this issue shortly.

7.5.2 Representation of Strong Entity Sets with Complex Attributes

When a strong entity set has nonsimple attributes, things are a bit more complex. We handle composite attributes by creating a separate attribute for each of the component attributes; we do not create a separate attribute for the composite attribute itself. To illustrate, consider the version of the *instructor* entity set depicted in Figure 7.11. For the composite attribute *name*, the schema generated for *instructor* contains the attributes *first_name*, *middle_name*, and *last_name*; there is no separate attribute or schema for *name*. Similarly, for the composite attribute *address*, the schema generated contains the attributes *street*, *city*, *state*, and *zip_code*. Since *street* is a composite attribute it is replaced by *street_number*, *street_name*, and *apt_number*. We revisit this matter in Section 8.2.

Multivalued attributes are treated differently from other attributes. We have seen that attributes in an E-R diagram generally map directly into attributes for the appropriate relation schemas. Multivalued attributes, however, are an exception; new relation schemas are created for these attributes, as we shall see shortly.

Derived attributes are not explicitly represented in the relational data model. However, they can be represented as "methods" in other data models such as the object-relational data model, which is described later in Chapter 22.

The relational schema derived from the version of entity set *instructor* with complex attributes, without including the multivalued attribute, is thus:

instructor (*ID*, *first_name*, *middle_name*, *last_name*,
street_number, *street_name*, *apt_number*,
city, *state*, *zip_code*, *date_of_birth*)

For a multivalued attribute M, we create a relation schema R with an attribute A that corresponds to M and attributes corresponding to the primary key of the entity set or relationship set of which M is an attribute.

As an illustration, consider the E-R diagram in Figure 7.11 that depicts the entity set *instructor*, which includes the multivalued attribute *phone_number*. The primary key of *instructor* is *ID*. For this multivalued attribute, we create a relation schema

instructor_phone (*ID*, *phone_number*)

Each phone number of an instructor is represented as a unique tuple in the relation on this schema. Thus, if we had an instructor with *ID* 22222, and phone numbers 555-1234 and 555-4321, the relation *instructor phone* would have two tuples (22222, 555-1234) and (22222, 555-4321).

We create a primary key of the relation schema consisting of all attributes of the schema. In the above example, the primary key consists of both attributes of the relation *instructor phone*.

The entity set can be represented by just the following schema created from the multivalued composite attribute:

time slot (*time slot id, day, start time, end time*)

Although not represented as a constraint on the E-R diagram, we know that there cannot be two meetings of a class that start at the same time of the same day-of- the-week but end at different times; based on this constraint, *end time* has been omitted from the primary key of the *time slot* schema.

7.5.3 Representation of Weak Entity Sets

Let A be a weak entity set with attributes $a_1, a_2, \ldots, a_m$. Let B be the strong entity set on which A depends. Let the primary key of B consist of attributes $b_1, b_2, \ldots, b_n$. We represent the entity set A by a relation schema called A with one attribute for each member of the set:

$$\{a_1, a_2, \ldots, a_m\} \cup \{b_1, b_2, \ldots, b_n\}$$

For schemas derived from a weak entity set, the combination of the pri- mary key of the strong entity set and the discriminator of the weak entity set serves as the primary key of the schema. In addition to creating a primary key, we also create a foreign-key constraint on the relation A, specifying that the attributes $b_1, b_2, \ldots, b_n$ reference the primary key of the relation B. The foreign- key constraint ensures that for each tuple representing a weak entity, there is a corresponding tuple representing the corresponding strong entity.

As an illustration, consider the weak entity set *section* in the E-R diagram of Figure 7.15. This entity set has the attributes: *sec id*, *semester*, and *year*. The primary key of the *course* entity set, on which *section* depends, is *course id*. Thus, we represent *section* by a schema with the following attributes:

section (*course id, sec id, semester, year*)

The primary key consists of the primary key of the entity set *course*, along with the discriminator of *section*, which is *sec id*, *semester*, and *year*.

7.5.4 Representation of Relationship Sets

Let R be a relationship set, let $a_1, a_2, \ldots, a_m$ be the set of attributes formed by the union of the primary keys of each of the entity sets participating in R, and let the descriptive attributes (if any) of R be $b_1, b_2, \ldots, b_n$. We represent this relationship set by a relation schema called R with one attribute for each member of the set:

$$\{a_1, a_2, \ldots, a_m\} \cup \{b_1, b_2, \ldots, b_n\}$$

We described earlier, in Section 7.3.3, how to choose a primary key for a binary relationship set. As we saw in that section, taking all the primary-key attributes from all the related entity sets serves to identify a particular tuple, but for one-to- one, many-to-one, and one-to-many relationship sets, this turns out to be a larger set of attributes than we need in the primary key. The primary key is instead chosen as follows:

- For a binary many-to-many relationship, the union of the primary-key at- tributes from the participating entity sets becomes the primary key.
- For a binary one-to-one relationship set, the primary key of either entity set can be chosen as the primary key. The choice can be made arbitrarily.

- For a binary many-to-one or one-to-many relationship set, the primary key of the entity set on the "many" side of the relationship set serves as the primary key.

For an *n*-ary relationship set without any arrows on its edges, the union of the primary key-attributes from the participating entity sets becomes the primary key.

- For an *n*-ary relationship set with an arrow on one of its edges, the primary keys of the entity sets not on the "arrow" side of the relationship set serve as the primary key for the schema. Recall that we allowed only one arrow out of a relationship set.

As an illustration, consider the relationship set *advisor* in the E-R diagram of Figure 7.15. This relationship set involves the following two entity sets:

- *instructor* with the primary key *ID*.
- *student* with the primary key *ID*.

7.6.4.1 Redundancy of Schemas

A relationship set linking a weak entity set to the corresponding strong entity set is treated specially. As we noted in Section 7.5.6, these relationships are many-to- one and have no descriptive attributes. Furthermore, the primary key of a weak entity set includes the primary key of the strong entity set. In the E-R diagram of Figure 7.14, the weak entity set *section* is dependent on the strong entity set *course* via the relationship set *sec course*. The primary key of *section* is *course id*, *sec id*, *semester*, *year* and the primary key of *course* is *course id*. Since *sec course* has no descriptive attributes, the *sec course* schema has attributes *course id*, *sec id*, *semester*, and *year*. The schema for the entity set *section* includes the attributes *course id*, *sec id*, *semester*, and *year* (among others). Every (*course id*, *sec id*, *semester*, *year*) combination in a *sec course* relation would also be present in the relation on schema *section*, and vice versa. Thus, the *sec course* schema is redundant.

In general, the schema for the relationship set linking a weak entity set to its corresponding strong entity set is redundant and does not need to be present in a relational database design based upon an E-R diagram.

7.6.4.2 Combination of Schemas

Consider a many-to-one relationship set *AB* from entity set *A* to entity set *B*. Using our relational-schema construction algorithm outlined previously, we get

three schemas: *A*, *B*, and *AB*. Suppose further that the participation of *A* in the relationship is total; that is, every entity *a* in the entity set *B* must participate in the relationship *AB*. Then we can combine the schemas *A* and *AB* to form a single schema consisting of the union of attributes of both schemas. The primary key of the combined schema is the primary key of the entity set into whose schema the relationship set schema was merged.

To illustrate, let's examine the various relations in the E-R diagram of Fig- ure 7.15 that satisfy the above criteria:

- *inst dept*. The schemas *instructor* and *department* correspond to the entity sets *A* and *B*, respectively. Thus, the schema *inst dept* can be combined with the *instructor* schema. The resulting *instructor* schema consists of the attributes
  {*ID*, *name*, *dept name*, *salary*}.
- *stud dept*. The schemas *student* and *department* correspond to the entity sets *A* and *B*, respectively. Thus, the schema *stud dept* can be combined with the *student* schema. The resulting *student* schema consists of the attributes {*ID*, *name*, *dept name*, *tot cred*}.
- *course dept*. The schemas *course* and *department* correspond to the entity sets *A* and *B*, respectively. Thus, the schema *course dept* can be combined with the *course* schema. The resulting *course* schema consists of the attributes {*course id*, *title*, *dept name*, *credits*}.
- *sec class*. The schemas *section* and *classroom* correspond to the entity sets *A* and *B*, respectively. Thus, the schema

sec class can be combined with the *section* schema. The resulting *section* schema consists of the attributes {*course id*, *sec id*, *semester*, *year*, *building*, *room number*}.

- *sec time slot*. The schemas *section* and *time slot* correspond to the entity sets *A* and *B* respectively, Thus, the schema *sec time slot* can be combined with the *section* schema obtained in the previous step. The resulting *section* schema consists of the attributes {*course id*, *sec id*, *semester*, *year*, *building*, *room number*, *time slot id*}.

In the case of one-to-one relationships, the relation schema for the relationship set can be combined with the schemas for either of the entity sets.

7.6 Entity-Relationship Design Issues

In this section, we examine basic issues in the design of an E-R database schema.

7.6.1 Use of Entity Sets versus Attributes

Consider the entity set *instructor* with the additional attribute *phone number* (Fig- ure 7.17a.) It can easily be argued that a phone is an entity in its own right with attributes *phone number* and *location*; the location may be the office or home where the phone is located, with mobile (cell) phones perhaps represented by the value "mobile." If we take this point of view, we do not add the attribute *phone number* to the *instructor*. Rather, we create:

- A *phone* entity set with attributes *phone number* and *location*.
- A relationship set *inst phone*, denoting the association between instructors and the phones that they have.

7.6.2 Use of Entity Sets versus Relationship Sets

It is not always clear whether an object is best expressed by an entity set or a relationship set. In Figure 7.15, we used the *takes* relationship set to model the situation where a student takes a (section of a) course. An alternative is to imagine ssthat there is a course-registration record for each course that each student takes. Then, we have an entity set to represent the course-registration record. Let us call that entity set *registration*. Each *registration* entity is related to exactly one student and to exactly one section, so we have two relationship sets, one to relate course- registration records to students and one to relate course-registration records to sections. In Figure 7.18, we show the entity sets *section* and *student* from Figure 7.15 with the *takes* relationship set replaced by one entity set and two relationship sets:

- *registration*, the entity set representing course-registration records.
- *section reg*, the relationship set relating *registration* and *course*.
- *student reg*, the relationship set relating *registration* and *student*.

Note that we use double lines to indicate total participation by *registration* entities.

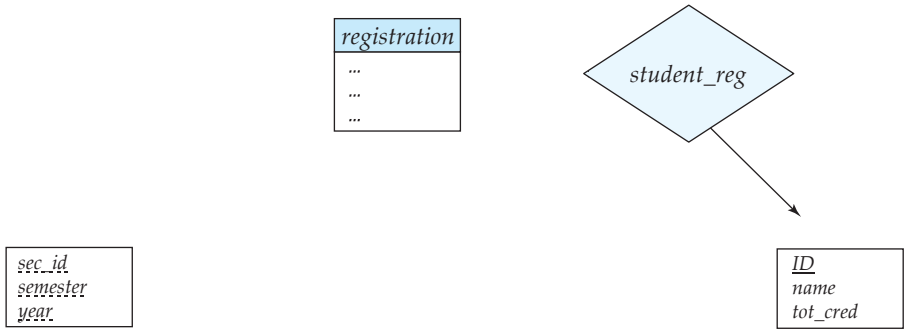


Figure 7.18 Replacement of *takes* by *registration* and two relationship sets

Both the approach of Figure 7.15 and that of Figure 7.18 accurately represent the university's information, but the use of *takes* is more compact and probably preferable. However, if the registrar's office associates other information with a course-registration record, it might be best to make it an entity in its own right.

One possible guideline in determining whether to use an entity set or a relationship set is to designate a relationship set to describe an action that occurs between entities. This approach can also be useful in deciding whether certain attributes may be more appropriately expressed as relationships.

7.6.3 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* provides us a record of 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.

In fact, it is always possible to replace a nonbinary (n-ary, for $n > 2$) relation- ship set by a number of distinct binary relationship sets. For simplicity, consider the abstract ternary (n 3) relationship set R, relating entity sets A, B, and C. We replace the relationship set R by an entity set E, and create three relationship sets as shown in Figure 7.19:

- R_A, relating E and A.
- R_B , relating E and B.
- R_C , relating E and C.

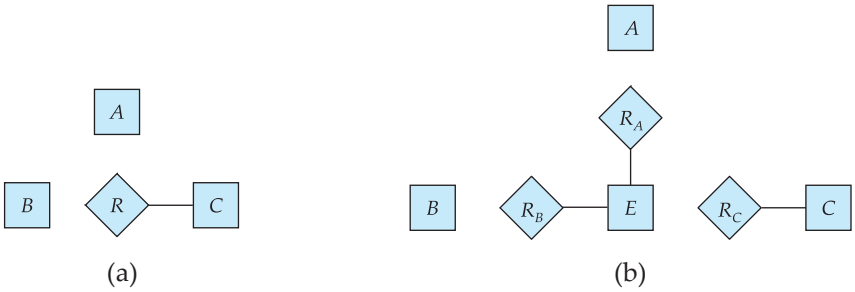


Figure 7.19 Ternary relationship versus three binary relationships.

If the relationship set R had any attributes, these are assigned to entity set E ; further, a special identifying attribute is created for E (since it must be possible to distinguish different entities in an entity set on the basis of their attribute values). For each relationship (a_i , b_i , c_i) in the relationship set R, we create a new entity e_i in the entity set E. Then, in each of the three new relationship sets, we insert a relationship as follows:

- (e_i , a_i) in R_A.
- (e_i , b_i) in R_B.
- (e_i , c_i) in R_C.

We can generalize this process in a straightforward manner to n-ary relation- ship sets. Thus, conceptually, we

can restrict the E-R model to include only binary relationship sets. However, this restriction is not always desirable.

- An identifying attribute may have to be created for the entity set created to represent the relationship set. This attribute, along with the extra relationship sets required, increases the complexity of the design and (as we shall see in Section 7.6) overall storage requirements.
- An n-ary relationship set shows more clearly that several entities participate in a single relationship.
- There may not be a way to translate constraints on the ternary relationship into constraints on the binary relationships. For example, consider a con- straint that says that R is many-to-one from A, B to C; that is, each pair of entities from A and B is associated with at most one C entity. This constraint cannot be expressed by using cardinality constraints on the relationship sets R_A, R_B, and R_C.

Consider the relationship set *proj guide* in Section 7.2.2, relating *instructor*, *student*, and *project*. We cannot directly split *proj guide* into binary relationships between *instructor* and *project* and between *instructor* and *student*. If we did so,

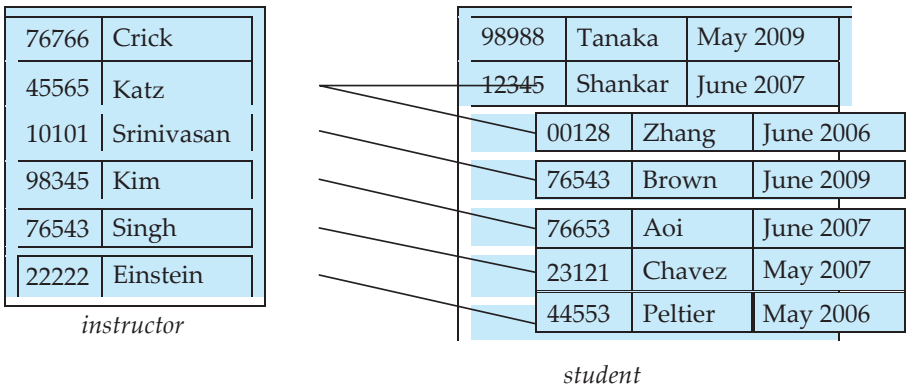


Figure 7.20 *date* as an attribute of the *student* entity set.

we would be able to record that instructor Katz works on projects A and B with students Shankar and Zhang; however, we would not be able to record that Katz works on project A with student Shankar and works on project B with student Zhang, but does not work on project A with Zhang or on project B with Shankar.

The relationship set *proj guide* can be split into binary relationships by creating a new entity set as described above. However, doing so would not be very natural.

7.6.4 Placement of Relationship Attributes

The cardinality ratio of a relationship can affect the placement of relationship attributes. Thus, attributes of one-to-one or one-to-many relationship sets can be associated with one of the participating entity sets, rather than with the relation- ship set. For instance, let us specify that *advisor* is a one-to-many relationship set such that one instructor may advise several students, but each student can be advised by only a single instructor. In this case, the attribute *date*, which specifies when the instructor became the advisor of a student, could be associated with the *student* entity set, as Figure 7.20 depicts. (To keep the figure simple, only some of the attributes of the two entity sets are shown.) Since each *student* entity participates in a relationship with at most one instance of *instructor*, making this attribute designation has the same meaning as would placing *date* with the *advisor* relationship set. Attributes of a one-to-many relationship set can be repositioned to only the entity set on the "many" side of the relationship. For one-to-one rela- tionship sets, on the other hand, the relationship attribute can be associated with either one of the participating entities.

NORMAL FORMS

Given a relation schema, we need to decide whether it is a good design or we need to decompose it into smaller relations. Such a decision must be guided by an understanding of what problems, if any, arise from the current schema.
To provide such guidance, several normal forms have been proposed. If a relation schema is in one of these normal forms, we know that certain kinds of problems cannot arise. The normal forms based on FDs are first normal form (1NF), second normal forM (2NF) , third norrnalform(3NF), and Boyce-Codd normal form (BCNF). These forms have increasingly restrictive requirements: Every relation in BCNF is also in 3NF, every relation in 3NF is also in 2NF, and every relation in 2NF is in 1NF.

4.0 Normalization of Database: Database Normalisation is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anamolies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.
Normalization is used for mainly two purpose,

- Eliminating reduntant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

4.1.2 Problem Without Normalization: Without Normalization, it becomes difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anamolies are very frequent if Database is not Normalized. To understand these anomalies let us take an example of **Student** table.

S_id	S_Name	S_Address	Subject_opted
401	Adam	Noida	Bio
402	Alex	Panipat	Maths
403	Stuart	Jammu	Maths
404	Adam	Noida	Physics

Figure 3.6 Student table

- **Updation Anamoly :** To update address of a student who occurs twice or more than twice in a table, we will have to update S_Address column in all the rows, else data will become inconsistent.
- **Insertion Anamoly :** Suppose for a new admission, we have a Student id(S_id), name and address of a student but if student has not opted for any subjects yet then we have to insert NULL there, leading to Insertion Anamoly.
- **Deletion Anamoly :** If (S_id) 401 has only one subject and temporarily he drops it, when we delete that row, entire student record will be deleted along with it.

4.1.3 Normalization Rule: Normalization rule are divided into following normal form.

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF
5. Multivalve dependency (4NF, 5NF)

First Normal Form(1NF): As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

Student	Age	Subject
Adam	15	Biology, Maths
Alex	14	Maths
Stuart	17	Maths

Figure 3.7: Student table

In First Normal Form, any row must not have a column in which more than one value is saved, like separated with commas. Rather than that, we must separate such data into multiple rows i.e the value should be automic in row and column intersection. Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Student	Age	Subject
Adam	15	Biology
Adam	15	Maths
Alex	14	Maths
Stuart	17	Maths

Figure 3.8 Student table in 1NF

Second Normal Form (2NF): As per the Second Normal Form there must ***not be any partial dependency*** of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails Second normal form.

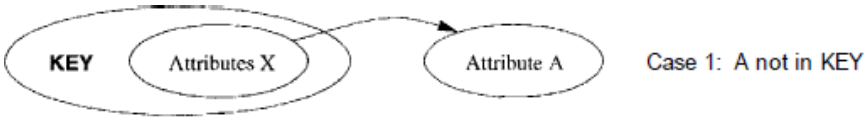


Figure Partial Dependencies

In example of First Normal Form there are two rows for Adam, to include multiple subjects that he has opted for. While this is searchable, and follows First normal form, it is an inefficient use of space. Also in the above Table in First Normal Form, while the candidate key is {Student, Subject}, Age of Student only depends on Student column, which is incorrect as per Second Normal Form. To achieve second normal form, it would be helpful to split out the subjects into an independent table, and match them up using the student names as foreign keys.

Student	Age
Adam	15
Alex	14
Stuart	17

Student	Subject
Adam	Biology
Adam	Maths
Alex	Maths
Stuart	Maths

Figure 3.9 New Student Table New Subject Table introduced for 2NF

In Student Table the candidate key will be Student column, because all other column i.e Age is dependent on it. In Subject Table the candidate key will be {Student, Subject} column. Now, both the above tables qualifies for Second Normal Form and will never suffer from Update Anomalies. Although there are a few complex cases in which table in Second Normal Form suffers Update Anomalies, and to handle those scenarios Third Normal Form is there.

Third Normal Form (3NF): Let R be a relation schema, F be the set of FDs given to hold over R, X be a subset of the attributes of R, and A be an attribute of R. R is in third normal form if, for every FD X -+ A in F, one of the following statements is true:

- A E X; that is, it is a trivial FD, or
- X is a super key, or
- A is part of some key for R.

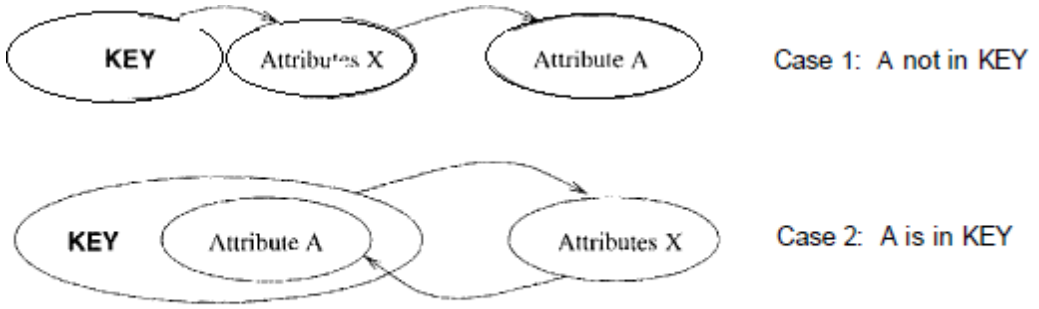


Figure Transitive Dependencies

Third Normal form applies that every non-prime attribute of table must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another non-prime attribute. So this ***transitive functional dependency should be removed*** from the table and also the table must be in Second Normal form. For example, consider a table with following fields.

Student_Detail Table :

Student_id	Student_name	DOB	Street	city	State	Zip

Figure 3.10 Student_Detail table Not in 3NF

In this table Student_id is Primary key, but street, city and state depends upon Zip. The dependency between zip and other fields is called ***transitive dependency***. Hence to apply 3NF, we need to move the street, city and state to new table, with Zip as primary key.

New Student_Detail Table :

Student_id	Student_name	DOB	Zip

Address Table :

Zip	Street	city	state

Figure 3.10 Table in 3NF

The advantage of removing ***transitive dependency*** is,

- Amount of data duplication is reduced.
- Data integrity achieved.

Boyce and Codd Normal Form (BCNF): Let R be a relation scherna, R be the set ofFD's given to hold over R. X be a subset of the attributes of R, and A be an attribute of R. R is in Boyce-Codd normal form if, for everyFD X $\in$ A in F, one of the follo\ving statements is true:

- A E X; that is, it is a trivial FD, or
- X is asuperkey.

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anamoly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and, for each functional dependency (X -> Y), X should be a super Key.

A relation is said to undergo BCNF normal form if it is in 3rd normal form and if the following systems are noticed.

- The relation will have multiple composite keys
- Further the composite keys will have common attribute
- And the key of first composite key is functionally dependent on key of other composite key.

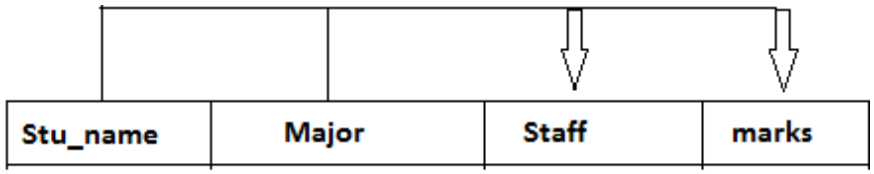
For example

Stu_name	Major	Staff	marks
Raju	Physics	Prof. John	80
Raju	Computers	David	97
Giri	Physics	Prof. John	77
Hari	Computers	David	86
Srinu	Physics	Prof. John	72

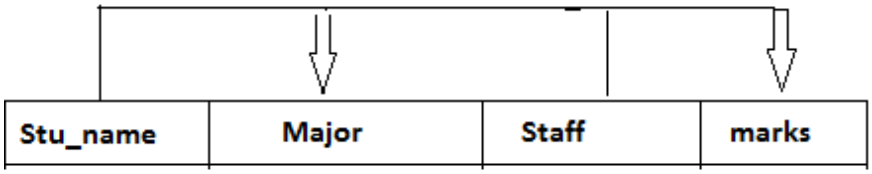
Figure 3.11 Student_Major table NOT in BCNF

In the above example the two composite keys are

1. Stu_name and Major
2. Stu_name and Staff



Stu_name and Major acts as Composit Key 1



Stu_name and Staff acts as Composit Key 2

Further we can notice that stu_name is the common attribute in both the keys.
Also it is noticed that there is functional dependency between Major and Staff. Whenever the major subject is Physics, it is dealt by staff Prof. John, Computers is dealt by Mr. David. So if many students takes major subject as Physics, in the corresponding column Prof. John will be repeatedly occurs and the same in the case of Computers. The redundancy can be avoided by applying the BCNF normalization techniques.

Stu_name	Major	marks
Raju	Physics	80
Raju	Computers	97
Giri	Physics	77
Hari	Computers	86
Srinu	Physics	72

Major	Staff
Physics	Prof. John
Computers	David

Figure 3.12 Table Student and Table Major after BCNF normalization

The relation is decomposed into two sub relations Student and Major tables with columns Student{stu_name, Major, marks} and Major{Major, staff}

Multivalued Dependency: The multivalued dependency X $\in\in$ Y holds in a relation R if for each value of X, there exists a definite set of values of Y. whenever we have two tuples of R that agree in all the attributes of X, then we can swap their Y components and get two new tuples that are also in R.

Formal definition of Multi valued dependency:

Let R be a relational schema and let $\alpha \subseteq R$ and $\beta \subseteq R$ (subsets). The multivalued dependency

$\alpha \twoheadrightarrow \beta$

(which can be read as α multidetermines β) holds on R if, in any legal relation $r(R)$,

r such that

for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

In more simple words the above condition can be expressed as follows: if we denote by (x, y, z) the tuple having values for α, β, $R - \alpha - \beta$ collectively equal to x, y, z, correspondingly,
then whenever the tuples (a, b, c) and (a, d, e) exist in r, the tuples (a, b, e) and (a, d, c) should also exist in r.

Consider this example of a relation of university courses, the books recommended for the course, and the lecturers who will be teaching the course:

University courses

Course	Book	Lecturer
AHA	Silberschatz	John D
AHA	Nederpelt	John D
AHA	Silberschatz	William M
AHA	Nederpelt	William M
AHA	Silberschatz	Christian G
AHA	Nederpelt	Christian G
OSO	Silberschatz	John D
OSO	Silberschatz	William M

Figure 3.13 University Courses with Multi valued dependency

Because the lecturers attached to the course and the books attached to the course are independent of each other, this database design has a multivalued dependency; if we were to add a new book to the AHA course, we would have to add one record for each of the lecturers on that course, and vice versa. Put formally, there are two multivalued dependencies in this relation: {course} €€ {book} and equivalently {course} €€ {lecturer}. Databases with multivalued dependencies thus exhibit redundancy. In database normalization, fourth normal form requires that either every multivalued dependency X €€ Y is trivial or for every nontrivial multivalued dependency X €€ Y, X is a super key. A multivalued dependency X €€ Y is trivial if Y is a subset of X, or if X **U** Y is the whole set of attributes of the relation.

4.1 PROPERTIES OF DECOMPOSITIONS

4.1.1 Lossless-Join Decomposition: Let R be a relation schema and let F be a, set of FDs over **R**. A decomposition of R into two schemas with attribute sets **X** and **Y** is said to be a lossless-join decomposition with respect to F if, for every instance T of R that satisfies the dependencies in **F, *π x(r) x π y(r)* = T**. In other words, we can recover the original relation from the decomposed relations.

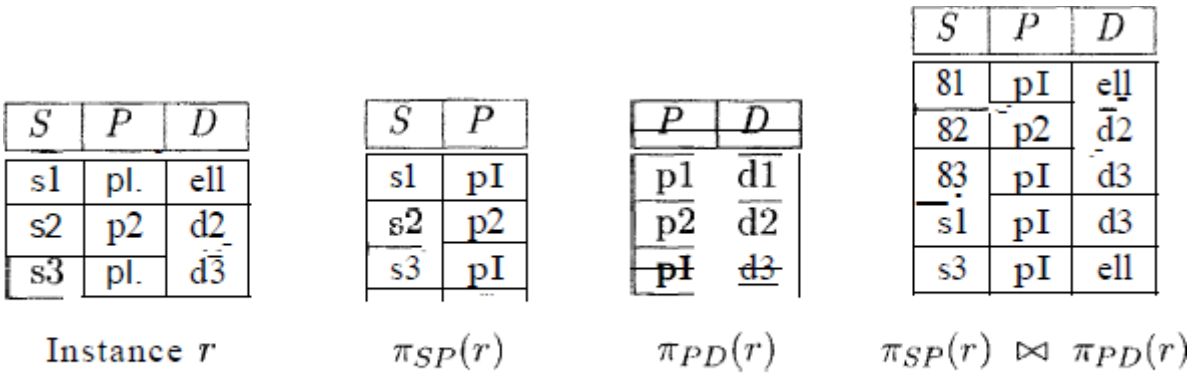


Figure 3.14 Instances Illustrating Lossy Decompositions

4.1.2 Dependency Preservation: A decomposition of a relation R into R1, R2, R3, …, Rn is dependency preserving decomposition with respect to the set of Functional Dependencies F that hold on R only if the following is hold;

(F1 U F2 U F3 U … U Fn)+ = F+
where,
F1, F2, F3, …, Fn – Sets of Functional dependencies of relations R1, R2, R3, …, Rn. (F1 U F2 U F3 U … U Fn)+ - Closure of Union of all sets of functional dependencies.

F+ - Closure of set of functional dependency F of R.
If the closure of set of functional dependencies of individual relations R1, R2, R3, …, Rn are equal to the set of functional dependencies of the main relation R (before decomposition), then we would say the decomposition D is lossless dependency preserving decomposition.
Few key points:

- We would like to check easily that updates to the database do not result in illegal relations being created.
- It would be nice if our design allowed us to check updates without having to compute natural joins.
- We can permit a non-dependency preserving decomposition if the database is static. That is, if there is no new insertion or update in the future.

Example:
Assume R(A, B, C, D) with FDs A→B, B→C, C→D.
Let us decompose R into R1 and R2 as follows;
R1(A, B, C)
R2(C, D)
The FDs A→B, and B→C are hold in R1.
The FD C→D holds in R2.
All the functional dependencies hold here. Hence, this decomposition is dependency preserving.

4.2 SCHEMA REFINEMENT IN DATABASE DESIGN

Database designers typically use a conceptual design methodology, such as ER design, to arrive at an initial database design. Given this, the approach redundancy can be eliminated and normaliazation can be attained with schema refinement by decomposition of relation.

4.2.1 Constraints on an Entity Set: Consider the Hourly_Emp relation again. The constraint that attribute ssn is a key can be expressed as FD
{ssn} $\in$ {ssn, name, lot, rating, hourly_wages, hours_worked }

For clarity, we write this FD as S$\in$SNLRWH, using a single letter to denote each attribute. In addition, the constraint that the hourly_wages attribute is determined by the rating attribute is an FD: R$\in$W

This leads to redundant storage of rating wagge associations. It cannot be expressed in terms of ER model. Only FDs that determine all attributes of a relation (key constraints) can be expressed in the ER model. Therefore, we could not detect it when we considered Hourly_Employees as an entity set during ER modeling.

To avoid the problem we can introduce an entity set called Wage_Table(with attributes rating and hourly_wage) and a relationship set Has_Wages associating Hourly_Employees and Wage_Table.

4.2.2 Constraints on a Relationship Set: Suppose that we have entity sets *Parts*, *Suppliers* and *Departments*, as well as a relationship set *Contracts* that involves all of them. We refer to the schema for contracts as CQPSD. A contract with contract id C specifies that a supplier S will supply some quantity Q of a part. P to a department D.

We assume a policy that a department purchases at most one part from any given supplier. Therefore, if there are several contracts between the same supplier and department, we know that the same part must be involved in all of them. This constraint is an FD, DS -> P.

Again we have redundancy and its associated problems. We can address this situation by decomposing Contracts into two relations with attributes CQSD and SDP. Intuitively, the relation SDP records the part supplied to a department by a supplier, and the relation CQSD records additional information about a contract. It is unlikely that we would arrive at such a design solely through ER modeling. Since it is hard to formulate an entity or relationship that corresponds naturally toCQSD.

4.2.3 Identifying Attributes of Entities: This example illustrates how a careful examination of FDs can lead to a better understanding of the entities and relationships underlying the relational tables; in particular, it shows that attributes can easily be associated with the 'wrong' entity set during ER design. The ER diagram in Figure 19.11 shows a relationship set called

Works_In that is similar to the Works_ln relationship set but with an additional key constraint indicating that an employee can work in at most one department.

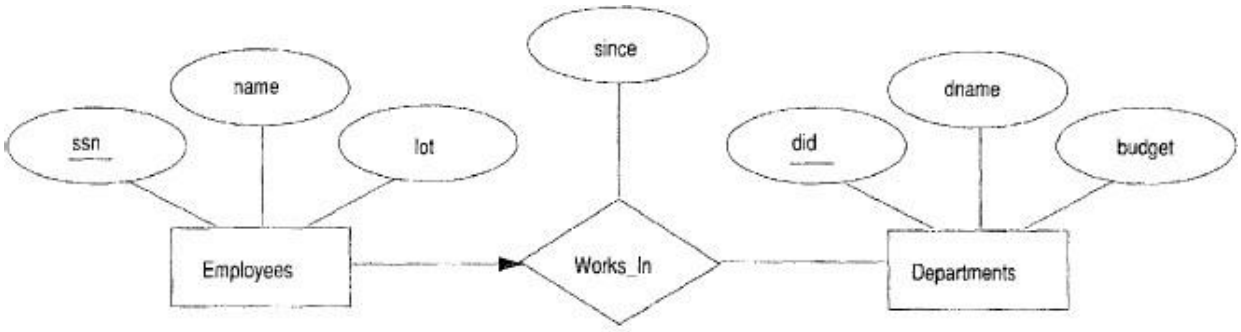


Figure 19.11. The Works_In Relationship Set

Using the key constraint, we can translate this ER diagram into two relations: Workers(ssn, name,
lot, did, since)
Departments(did, dname, budget)
The entity set Employees and the relationship set Works_in are mapped to a single relation, Workers.

Now suppose employees are assigned parking lots based on their department, and that all employees in a given department are assigned to the same lot. This constraint is not expressible with respect to the ER, diagrarm of Figure 19.11. It is another example of an FD: did $\in$ lot. The redundancy in this design can be eliminated by decomposing the Workers relation into two relations:

Workers2(ssn, name, did, since)
Dept_Lots(did, lot)

The new design can be as follows.

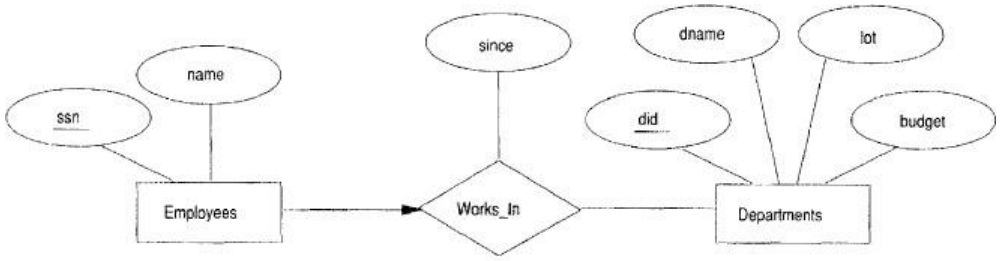


Figure 19.12 Refined\Norks_In Relationship Set

4.2.4 Identifying Entity Sets: Consider a variant of the Reserves schema used earlier. Let Reserves contain attributes S, B, and D as before, indicating that sailor S has a reservation for boat B on day D. In addition, let there be an attribute G denoting the credit card to which the reservation is charged. We use this example to illustrate how FD information can be used to refine an ER design. In particular, we discuss how FD information can help decide whether a concept should be modeled as an entity or as an attribute.

Suppose every sailor uses a unique credit card for reservations. This constraint is expressed by the FD S→C. This constraint indicates that, in relation Reserves, we store the credit card number for a sailor as often as we have reservations for that sailor, and we have redundancy and potential update anomalies. A solution is to decompose Reserves into two relations with attributes SBD and SC. intuitively, one holds information about reservations, and the other holds information about credit cards.

It is instructive to think about an ER design that would lead to these relations. One approach is to introduce an entity set called Credit_Cards, with the sale attribute cardno, and a relationship set *Has_Card* associating Sailors and *Credit_Cards*. By noting that each credit card belongs to a single sailor, we can map *Has_Card* and *Credit_Cards* to a single relation with attributes SC. We would probably not model credit card numbers as entities if our main interest in card numbers is to indicate how a reservation is to be paid for; it suffices to use an attribute to model card numbers in this situation.

A second approach is to make *cardno* an attribute of *Sailors*. But this approach is not very natural-sailor may have several cards, and we are not interested in all of them. Our interest is in the one card that is used to pay for reservations, which is best modeled as an attribute of the relationship *Iteserves*. A helpful way to think about the design problem in this example is that we first make *cardno* an attribute of Reserves and then refine the resulting tables by taking into account the FD information.

4.3 Other kinds of dependencies: FDs are probably the most common and important kind of constraint from the point of database design. However, there are several other kinds of dependencies.

In particular, there is a well developed theory of database design using multi valued dependencies. By taking such dependencies into account, we can identify potential redundancy problems that cannot be detected using FDs alone.

4.3.1 Multivalued Dependencies: Suppose that we have a relation with attributes course, teacher, and book, which we denote as CTB. The meaning of a tuple is that teacher T can teach course C, and book B is a reccnnmended text for the course. There are no FDs; the key is CTB. However, the recommended texts for a course are independent of the instructor. The instance shown in Figure 19.13 illustrates this situation.

course	teacher	book
Physics101	Green	Mechanics
PhysicslOl	Green	Optics
PhysicslOl	Brown	Mechanics
Physics101	Brown	Optics
Math301	Green	Mechanics
Math301	Green	Vectors
Math301	Green	Geometry

Figure 19.13 BCNF Relation with Redundancy That Is Revealed by MVDs

Note three points here:

1. The relation schema CTB is in BCNF; therefore we would not consider decomposing it further if we looked only at the FDs that hold over (CTB).
2. There is redundancy. The fact that Green can teach Physics101 is recorded once per recommended text for the course. Sirnilarly, the fact that Optics is a text for Physics101 is recorded once per potential teacher.
3. The redundancy can be eliminated by decomposing CTB into CT and CB

The redundancy in this example is due to the constraint that the texts for a course are independent of the instructors, which cannot be expressed in terms of FDs.

This constraint is an example of a multivalued dependency. Ideally, we should model this situation using two binary relationship sets, Instructors with attributes CT and Text with attributes CB. Because these are two essentially independent relationships, modeling them with a single ternary relationship set with attributes CTB is inappropriate.

Let R be a relation schema and let X and Y be subsets of the attributes of R. Intuitively, the multivalued dependency X $\in\in$ Y is said to hold over R if, in every legal instance r of R, each X value is associated with a set of Y values and this set is independent of the values in the other attributes.

Formally, if the MVD X $\in\in$ Y holds over R and Z = R - XY, the following must be true for every legal instance r of R

If t1 E r, t2 E r and tl.X == t2.X, then there must be some t3 E r such that tl.XY = t3.XY and t2.Z = t3.Z

Figure 19.14 illustrates this definition. If we are given the first two tuples and told that the MVD X $\in\in$ Y holds over this relation, we can infer that the relation instance must also contain the third tuple. Indeed, by interchanging the roles of the first two tuples treating the first tuple as t2 and the second tuple as t1--we can deduce that the tuple t4 must also be in the relation instance.

X	Y	Z	
a	b_1	c1	— tuple $t1$
a	b_2	c2	— tuple $t2$
a	b1	c2	— tuple t_3
a	b_2	c1	— tuple $t4$

Figure 19.14 Illustration of MVD Definition

- MVD Complementation: If $X \twoheadrightarrow Y$, then $X \twoheadrightarrow R - XY$.
- MVD .Augmentation: If $X \twoheadrightarrow$ Yand $W \supseteq Z$, then $WX \twoheadrightarrow YZ$.
- MVD Transitivity: If $X \twoheadrightarrow$ Yand $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow (Z - Y)$.

As an exanlple of the use of these rules, since we have $C \twoheadrightarrow T$ over CTB, MVD complelnentation allows us to infer that $C \twoheadrightarrow OTB - CT$ as well, that is, $C \twoheadrightarrow B$. The remaining two rules relate FDs and MVDs:

- Replication: If $X \rightarrow$ Y, then $X \twoheadrightarrow Y$.
- Coalescence: If $X \twoheadrightarrow$ Yand there is a W such that $W \cap Y$ is elnpty, $W \rightarrow Z$, and $Y \supseteq Z$, then $X \rightarrow Z$.

4.3.2 Fourth Normal Form:

Fourth Normal form is a direct generalization of BCNF. Let R be a relation schema, X and Y be nonempty subsets of the attributes of R, and F' be a set of dependencies that includes both FDs and MVDs. R is said to be in fourth normal form (4NF), if, for every MVD X€€Y that holds over R, one of the following statements is true:

- $Y \subseteq X$ or $XY = R$, or
- X is a superkey.

In reading this definition, it is important to understand that the definition of a key has not changed, the *key* must uniquely determine all attributes through FDs alone. X €€ Y is a MVD if Y X R or XY = R; such that MVD always hold.

The relation CTB is not in 4NF because C €€ T is a nontrivial MVD and C is not a key. We can eliminate the resulting redundancy by decomposing CTB into CT and CB; each of these relations is then in 4NF.

To use MVD information fully, we must understand the theory of MVDs. However, the following result due to Date and Fagin identifies conditions-detected using only FD information!~-under which we can safely ignore MVD information.

Using MVD information in addition to the FD infornlation will not reveal any redundancy. Therefore, if these conditions hold, we do not even need to identify all MVDs. If a relation schema is in BCNF, and at least one of its keys consists of a single attribute, it is also in 4NF.

An im.portant assumption is implicit in any application of the preceding result: *The set of FDs identified thus far is 'indeed the set of all FDs that hold over the relation.* This assulllption is important because the result relies on the relation being in BCNF, which in turn depends on the set of FDs that hold over the relation.

We illustrate this point using an example. Consider a relation schema ABCD and suppose that the FD A $\in$ BCD and the MVD B $\in\in$ C are given. Considering only these dependencies, the relation schema appears to be a counter example to the result. The relation has a simple key, appears to be in BCNF, and yet is not in 4NF because B $\in\in$ C: causes a violation of the 4NF conditions. Let us take a closer look.

B	C	A	D	
b	c1	a1	d_1	tuple t_1
b	c2	a2	d_2	tuple t2
b	c1	a2	d2	tuple t_3

Figure 19.15 Three Tuples form a Legal Instance of *ABCD*

Figure 19.15 shows three tuples from an instance of ABCD that satisfies the given MVD B $\in\in$ C. From the definition of an MVD given tuples t1 and t2 it follows that tuple t3 must also be included in the instance. Consider tuples t2 and t3, from the given FD A $\in$ BCD and the fact that these tuples have the same A value we can deduce that C1 = C2. Therefore we see that the FD B $\in$ C must hold over ABCD whenever the FD A $\in$ BCD and the MVD B $\in\in$ C hold. If B $\in$ C holds, the relation ABCD is not in BCNF (unless additional FDs make B a key)