11 Design and Dependencies

When the only tool you have is a hammer,
everything begins to look like a nail.
—Anonymous

Alice: *Will we use a hammer for schema design?*
Riccardo: *Sure: decomposition, semantic modeling, . . .*
Vittorio: *And each provides nails to which the data must fit.*
Sergio: *The more intricate the hammer, the more intricate the nail.*

We have discussed earlier applications of dependencies in connection with query optimization (Section 8.4) and user views (Section 10.2). In this chapter, we briefly consider how dependencies are used in connection with the design of relational database schemas.

The problem of designing database schemas is complex and spans the areas of cognitive science, knowledge representation, software practices, implementation issues, and theoretical considerations. Due to the interaction of these many aspects (some of them integrally related to how people think and perceive the world), we can only expect a relatively narrow and somewhat simplistic contribution from theoretical techniques. As a result, the primary focus of this chapter is to introduce the kinds of formal tools that are used in the design process; a broader discussion of how to use these tools in practice is not attempted. The interested reader is referred to the Bibliographic Notes, which indicate where more broad-based treatments of relational schema design can be found.

In the following discussion, designing a relational schema means coming up with a "good" way of grouping the attributes of interest into tables, yielding a database schema. The choice of a schema is guided by semantic information about the application data provided by the designer. There are two main ways to do this, and each leads to a different approach to schema design.

Semantic data model: In this approach (Section 11.1), the application data is first described using a model with richer semantic constructs than relations. Such models are called "semantic data models." The schema in the richer model is then translated into a relational schema. The hope is that the use of semantic constructs will naturally lead to specifying good schemas.

Refinement of relational schema: This approach (Section 11.2) starts by specifying an initial relational schema, augmented with dependencies (typically fd's and mvd's). The design process uses the dependencies to improve the schema. But what is it that makes

one schema better than another? This is captured by the notion of "normal form" for relational schemas, a central notion in design theory.

Both of these approaches focus on the transformation of a schema $\mathbf{S}_1$ into a relational schema $\mathbf{S}_2$. Speaking in broad terms, three criteria are used to evaluate the result of this transformation:

(1) Preservation of data;

(2) Desirable properties of $\mathbf{S}_2$, typically described using normal forms; and

(3) Preservation of "meta-data" (i.e., information captured by schema and dependencies).

Condition (1) requires that information not be lost when instances of $\mathbf{S}_1$ are represented in $\mathbf{S}_2$. This is usually formalized by requiring that there be a "natural" mapping $\tau : \mathit{Inst}(\mathbf{S}_1) \rightarrow \mathit{Inst}(\mathbf{S}_2)$ that is one-to-one. As we shall see, the notion of "natural" can vary, depending on the data model used for $\mathbf{S}_1$.

Criterion (2) has been the focus of considerable research, especially in connection with the approach based on refining relational schemas. In this context, the notion of relational schema is generalized to incorporate dependencies, as follows: A *relation schema* is a pair (R, Σ), where R is a relation name and Σ is a set of dependencies over R. Similarly, a *database schema* is a pair $(\mathbf{R}, \Sigma)$, where $\mathbf{R}$ is a database schema as before, and Σ is a set of dependencies over $\mathbf{R}$. Some of these may be *tagged* by a single relation (i.e., have the form $R_j : \sigma$, where σ is a dependency over $R_j \in \mathbf{R}$). Others, such as ind's, may involve pairs of relations. More generally, some dependencies might range over the full set of attributes occurring in $\mathbf{R}$. (This requires a generalization of the notion of dependency satisfaction, which is discussed in Section 11.3.)

With this notation established, we return to criterion (2). In determining whether one relational schema is better than another, the main factors that have been considered are redundancy in the representation of data and update anomalies. Recall that these were illustrated in Section 8.1, using the relations *Movies* and *Showings*. We concluded there that certain schemas yielded undesirable behavior. This resulted from the nature of the information contained in the database, as specified by a set of dependencies.

Although the dependencies are in some sense the cause of the problems, they also suggest ways to eliminate them. For example, the fd

$$\mathit{Movies}: \mathit{Title} \rightarrow \mathit{Director}$$

suggests that the attribute *Director* is a characteristic of *Title*, so the two attributes belong together and can safely be represented in isolation from the other data. It should be clear that one always needs some form of semantic information to guide schema design; in the absence of such information, one cannot distinguish "good" schemas from "bad" ones (except for trivial cases). As will be seen, the notion of normal form captures some characteristics of "good" schemas by guaranteeing that certain kinds of redundancies and update anomalies will not occur. It will also be seen that the semantic data model approach to schema design can lead to relational schemas in normal form.

In broad terms, the intuition behind criterion (3) is that properties of data captured by schema $\mathbf{S}_1$ (e.g., functional or inclusion relationships) should also be captured by schema $\mathbf{S}_2$. In the context of refining relational schemas, a precise meaning will be given for this criterion in terms of "preservation" of dependencies. We shall see that there is a kind of trade-off between criteria (2) and (3).

The approach of refining relational schemas typically makes a simplifying assumption called the "pure universal relation assumption" (pure URA). Intuitively, this states that the input schema $\mathbf{S}_1$ consists of a single relation schema, possibly with some dependencies. Section 11.3 briefly considers this assumption in a more general light. In addition, the "weak" URA is introduced, and the notions of dependency satisfaction and query interpretation are extended to this context.

This chapter is more in the form of a survey than the previous chapters, for several reasons. As noted earlier, more broad-based treatments of relational schema design may be found elsewhere and require a variety of tools complementary to formal analysis. The tools presented here can at best provide only part of the skeleton of a design methodology for relational schemas. Normal forms and the universal relation assumption were active research topics in the 1970s and early 1980s and generated a large body of results. Some of that work is now considered somewhat unfashionable, primarily due to the emergence of new data models. However, we mention these topics briefly because (1) they lead to interesting theoretical issues, and (2) we are never secure from a change of fashion.

11.1 Semantic Data Models

In this section we introduce semantic data models and describe how they are used in relational database design. Semantic data models provide a framework for specifying database schemas that is considerably richer than the relational model. In particular, semantic models are arguably closer than the relational model to ways that humans organize information in their own thinking. The semantic data models are precursors of the recently emerging *object-oriented database models* (presented in a more formal fashion in Chapter 21) and are thus of interest in their own right.

As a vehicle for our discussion, we present a semantic data model, called loosely the *generic semantic model* (GSM). (This is essentially a subset of the IFO model, one of the first semantic models defined in a formal fashion.) We then illustrate how schemas from this model can be translated into relational schemas. Our primary intention is to present the basic flavor of the semantic data model approach to relational schema design and some formal results that can be obtained. The presentation itself is somewhat informal so that the notation does not become overly burdensome.

In many practical contexts, the semantic model used is the *Entity-Relationship model* (ER model) or one of its many variants. The ER model is arguably the first semantic data model that appeared in the literature. We use the GSM because it incorporates several features of the semantic modeling literature not present in the ER model, and because the GSM presents a style closer to object-oriented database models.

GSM Schemas

Figure 11.1 shows the schema **CINEMA-SEM** from the GSM, which can be used to represent information on movies and theaters. The major building blocks of such schemas are abstract classes, attributes, complex value classes, and the ISA hierarchy; these will be considered briefly in turn.

The schema of Fig. 11.1 shows five classes that hold *abstract* objects: *Person*, *Director*, *Actor*, *Movie*, and *Theater*. These correspond to collections of similar objects in the world. There are two kinds of abstract class: *primary* classes, shown using diamonds, and *subclasses* shown using circles. This distinction will be clarified further when ISA relationships are discussed.

Instances of semantic schemas are constructed from the usual *printable* classes (e.g., **string**, **integer**, **float**, etc.) and "abstract" classes. The printable classes correspond to (subsets of) the domain **dom** used in the relational model. The printable classes are indicated using squares; in Fig. 11.1 we have labeled these to indicate the kind of values that populate them. Conceptually, the elements of an abstract class such as *Person* are actual persons in the world; in the formal model internal representations for persons are used. These internal representations have come to be known as *object identifiers* (OIDs). Because they are internal, it is usually assumed that OIDs cannot be presented explicitly to users, although programming and query languages can use variables that hold OIDs. The notion of instance will be defined more completely later and is illustrated in Example 11.1.1 and Fig. 11.2.

Attributes provide one mechanism for representing relationships between objects and other objects or printable values; they are drawn using arrows. For example, the *Person* class has attributes *name* and *citizenship*, which associate strings with each person object. These are examples of *single-valued* attributes. (In this schema, all attributes are assumed to be total.) *Multivalued* attributes are also allowed; these map each object to a set of objects or printable values and are denoted using arrows with double heads. For example, *acts_in* maps actors to the movies that they have acted in. It is common to permit *inverse* constraints between pairs of attributes. For example, consider the relationship between actors and movies. It can be represented using the multivalued attribute *acts_in* on *Actor* or the multivalued attribute *actors* on *Movie*. In this schema, we assume that the attributes *acts_in* and *actors* are constrained to be inverses of each other, in the sense that $m \in acts\_in(a)$ iff $a \in actor(m)$. A similar constraint is assumed between the attributes associating movies with directors.

In the schema **CINEMA-SEM**, the *Pariscope* node is an example of a *complex value class*. Members of the underlying class are triples whose coordinates are from the classes *Theater*, Time, and *Movie*, respectively. In the GSM, each complex value is the result of one application of the tuple construct. This is indicated using a node of the form $\otimes$, with components indicated using dashed arrows. The components of each complex value can be printable, abstract, or complex values. However, there cannot be a directed cycle in the set of edges used to define the complex values. As suggested by the attribute *price*, a complex value class may have attributes. Complex value classes can also serve as the range of an attribute, as illustrated by the class *Award*.

Complex values are of independent interest and are discussed in some depth in Chapter 20. Complex values generally include hierarchical structures built from a handful of

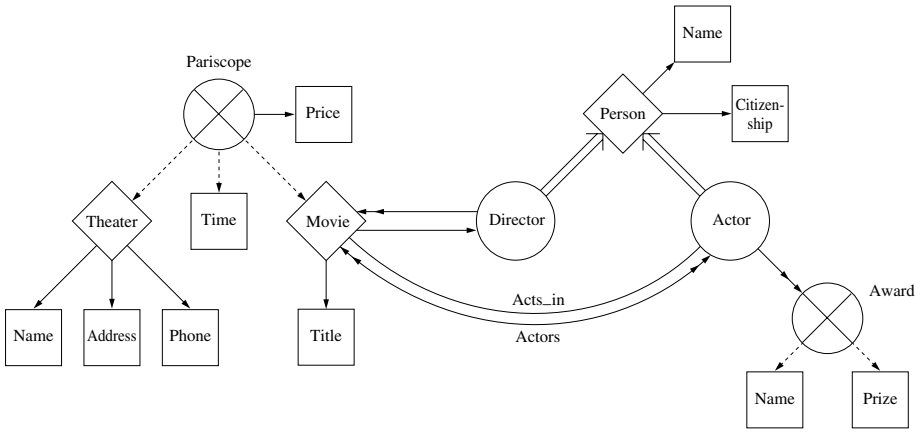

Figure 11.1: The schema **CINEMA-SEM** in the Generic Semantic Model

basic constructors, including tuple (as shown here) set, and sometimes others such as bag and list. Rich complex value models are generally incorporated into object-oriented data models and into some semantic data models. Some constructs for complex values, such as set, cannot be simulated directly using the pure relational model (see Exercise 11.24).

The final building block of the GSM is the *ISA* relationship, which represents set inclusion. In the example schema of Fig. 11.1, the ISA relationships are depicted by double-shafted arrows and indicate that the set of *Director* is a subset of *Person*, and likewise that *Actor* is a subset of *Person*. In addition to indicating set inclusion, ISA relationships indicate a form of subtyping relationship, or *inheritance*. Specifically, if class B ISA class A, then each attribute of A is also relevant (and defined for) elements of class B. In the context of semantic models, this should be no surprise because the elements of B are elements of A.

In the GSM, the graph induced by ISA relationships is a directed acyclic graph (DAG). The root nodes are primary abstract classes (represented with diamonds), and all other nodes are subclass nodes (represented with circles). Each subclass node has exactly one primary node above it. Complex value classes cannot participate in ISA relationships.

In the GSM, the tuple and multivalued attribute constructs are somewhat redundant: A multivalued attribute is easily simulated using a tuple construct. Such redundancy is typical of semantic models: The emphasis is on allowing schemas that correspond closely to the way that users think about an application. On a bit of a tangent, we also note that the tuple construct of GSM is close to the relationship construct of the ER model.

GSM Instances

Let **S** be a GSM schema. It is assumed that a fixed (finite or infinite) *domain* is associated to each printable class in **S**. We also assume a countably infinite set **obj** of OIDs.

An *instance* of **S** is a function **I** whose domain is the set of primary, subclass, and complex value classes of **S** and the set of attributes of **S**. For primary class C, $\mathbf{I}(C)$ is a finite set of OIDs, disjoint from $\mathbf{I}(C')$ for each other primary class C'. For each subclass D, $\mathbf{I}(D)$ is a set of OIDs, such that the inclusions indicated by the ISA relationships of **S** are satisfied. For complex value class C with components $D_1, \ldots, D_n$, $\mathbf{I}(C)$ is a finite set of tuples $\langle d_1, \ldots, d_n \rangle$, where $d_i \in \mathbf{I}(D_i)$ if D_i is an abstract or complex value class, and d_i is in the domain of D_i if D_i is a printable class. For a single-valued attribute f from C to C', $\mathbf{I}(f)$ is a function from $\mathbf{I}(C)$ to $\mathbf{I}(C')$ (or to the domain of C', if C' is printable). For a multivalued attribute f from C to C', $\mathbf{I}(f)$ is a function from $\mathbf{I}(C)$ to finite subsets of $\mathbf{I}(C')$ (or the domain of C', if C' is printable). Given instance **I**, attribute f from C to C', and object o in $\mathbf{I}(C)$, we often write $f(o)$ to denote $[\mathbf{I}(f)](o)$.

Example 11.1.1 Part of a very small instance $\mathbf{I}_1$ of **CINEMA-SEM** is shown in Fig. 11.2. The values of complex value *Award*, the attributes *award*, *address*, and *phone* are not shown. The symbols o_1, o_2, etc., denote OIDs.

Consider an instance $\mathbf{I}'$ that is identical to $\mathbf{I}_1$, except that o_2 is replaced by o_8 everywhere. Because OIDs serve only as internal representations that cannot be accessed

$\mathbf{I}_1(Person) =$ $\{o_1, o_2, o_3\}$	$name(o_1) = Alice$	$citizenship(o_1) = Great\ Britain$
	$name(o_2) = Allen$	$citizenship(o_2) = United\ States$
	$name(o_3) = Keaton$	$citizenship(o_3) = United\ States$
$\mathbf{I}_1(Director) = \{o_2\}$	$directed(o_2) = \{o_4, o_5\}$	
$\mathbf{I}_1(Actor) = \{o_2, o_3\}$	$acts\_in(o_2) = \{o_4, o_5\}$	
	$acts\_in(o_3) = \{o_5\}$	
$\mathbf{I}_1(Movie) = \{o_4, o_5\}$	$title(o_4) = Take\ the\ Money$ $and\ Run$	
	$title(o_5) = Annie\ Hall$	
	$director(o_4) = o_2$	$actors(o_4) = \{o_2\}$
	$director(o_5) = o_2$	$actors(o_5) = \{o_2, o_3\}$
$\mathbf{I}_1(Theater) = \{o_6\}$	$name(o_6) = Le\ Champo$	
$\mathbf{I}_1(Pariscope) =$ $\{\langle o_6, 20{:}00, o_4\rangle\}$	$price(\langle o_6, 20{:}00, o_4\rangle) = 30\text{FF}$	

Figure 11.2: Part of an instance $\mathbf{I}_1$ of **CINEMA-SEM**

explicitly, $\mathbf{I}_1$ and $\mathbf{I}'$ are considered to be identical in terms of the information that they represent.

Let **S** be a GSM schema. An *OID isomorphism* is a function μ that is a permutation on the set **obj** of OIDs and leaves all printables fixed. Such functions are extended to $Inst(\mathbf{S})$ in the natural fashion. Two instances $\mathbf{I}$ and $\mathbf{I}'$ are *OID equivalent*, denoted $\mathbf{I} \equiv_{OID} \mathbf{I}'$, if there is an OID isomorphism μ such that $\mu(\mathbf{I}) = \mathbf{I}'$. This is clearly an equivalence relation. As suggested by the preceding example, if two instances are OID equivalent, then they represent the same information. The formalism of OID equivalence will be used later when we discuss the relational simulation of GSM.

The GSM is a very basic semantic data model, and many variations on the semantic constructs included in the GSM have been explored in the literature. For example, a variety of simple constraints can be incorporated, such as cardinality constraints on attributes and disjointness between subclasses (e.g., that *Director* and *Actor* are disjoint). Another variation is to require that a class be "dependent" on an attribute (e.g., that each *Award* object must occur in the image of some *Actor*) or on a complex value class. More complex constraints based on first-order sentences have also been explored. Some semantic models support different kinds of ISA relationships, and some provide "derived data" (i.e., a form of user view incorporated into the base schema).

Translating into the Relational Model

We now describe an approach for translating semantic schemas into relational database schemas. As we shall see, the semantics associated with the semantic schema will yield dependencies of various forms in the relational schema.

A minor problem to be surmounted is that in a semantic model, real-world objects such as persons can be represented using OIDs, but printable classes must be used in the pure relational model. To resolve this, we assume that each primary abstract class has a *key*, that is, a set $\{k_1, \ldots, k_n\}$ of one or more attributes with printable range such that for each instance $\mathbf{I}$ and pair o, o' of objects in the class, $o = o'$ iff $k_1(o) = k_1(o')$ and ... and $k_n(o) = k_n(o')$. (Although more than one key might exist for a primary class, we assume that a single key is chosen.) In the schema **CINEMA-SEM**, we assume that (*person_*) *name* is the key for *Person*, that *title* is the key for *Movie*, and that (*theater_*)*name* is the key for *Theater*. (Generalizations of this approach permit the composition of attributes to serve as part of a key; e.g., including in the key for *Movie* the composition *director* $\circ$ *name*, which would give the name of the director of the movie.)

An alternative to the use of keys as just described is to permit the use of surrogates. Informally, a *surrogate* of an object is a unique, unchanging printable value that is associated with the object. Many real-world objects have natural surrogates (e.g., Social Security number for persons in the United States or France; or Invoice Number for invoices in a commercial enterprise). In other cases, abstract surrogates can be used.

The kernel of the translation of GSM schemas into relational ones concerns how objects in GSM instances can be represented using (tuples of) printables. For each class C occurring in the GSM schema, we associate a set of relational attributes, called the *representation* of C, and denoted $rep(C)$. For a printable class C, $rep(C)$ is a single attribute having this sort. For abstract class C, $rep(C)$ is a set of attributes corresponding to the key attributes of the primary class above C. For a complex value class $C = [C_1, \ldots, C_m]$, $rep(C)$ consists of (disjoint copies of) all of the attributes occurring in $rep(C_1), \ldots, rep(C_m)$.

Translation of a GSM schema into a relation schema is illustrated in the following example.

Example 11.1.2 One way to simulate schema **CINEMA-SEM** in the relational model is to use the schema **CINEMA-REL**, which has the following schema:

Person	[*name*, *citizenship*]
Director	[*name*]
Actor	[*name*]
Acts_in	[*name*, *title*]
Award	[*prize*, *year*]
Has_Award	[*name*, *prize*, *year*]
Movie	[*title*, *director_name*]
Theater	[*theater_name*, *address*, *phone*]
Pariscope	[*theater_name*, *time*, *title*, *price*]

Person	*name*	*citizenship*
	Alice	*Great Britain*
	Allen	*United States*
	Keaton	*United States*

Movie	*title*	*director_name*
	Take the Money and Run	*Allen*
	Annie Hall	*Allen*

Pariscope	*theater_name*	*time*	*title*	*price*
	Le Champo	20:00	*Take the Money and Run*	30FF

Figure 11.3: Part of a relational instance $\mathbf{I}_2$ that simulates $\mathbf{I}_1$

Figure 11.3 shows three relations in the relational simulation $\mathbf{I}_2$ of the instance $\mathbf{I}_1$ of Fig. 11.2.

In schema **CINEMA-REL**, both *Actor* and *Acts_in* are included in case there are one or more actors that did not act in any movie. For similar reasons, *Acts_in* and *Has_Award* are separated.

In contrast, we have assumed that each person has a citizenship (i.e., that citizenship is a total function). If not, then two relations would be needed in place of *Person*. Analogous remarks hold for directors, movies, theaters, and *Pariscope* objects.

In schema **CINEMA-REL**, we have not explicitly provided relations to represent the attributes *directed* of *Director* or *actors* of *Movie*. This is because both of these are inverses of other attributes, which are represented explicitly (by *Movie* and *Acts_in*, respectively).

If we were to consider the complex value class *Awards* of **CINEMA-SEM** to be dependent on the attribute *award*, then the relation *Award* could be omitted.

Suppose that $\mathbf{I}$ is an instance of **CINEMA-SEM** and that $\mathbf{I}'$ is the simulation of $\mathbf{I}$. The semantics of **CINEMA-SEM**, along with the assumed keys, imply that $\mathbf{I}'$ will satisfy several dependencies. This includes the following fd's (in fact, key dependencies):

$$
\begin{array}{lcl}
\mathit{Person} & : & \mathit{name} \rightarrow \mathit{citizenship} \\
\mathit{Movie} & : & \mathit{title} \rightarrow \mathit{director\_name} \\
\mathit{Theater} & : & \mathit{theater\_name} \rightarrow \mathit{address}, \mathit{phone} \\
\mathit{Pariscope} & : & \mathit{theater\_name}, \mathit{time}, \mathit{title} \rightarrow \mathit{price}
\end{array}
$$

A number of ind's are also implied:

$$
\begin{array}{lcl}
\mathit{Director}[\mathit{name}] & \subseteq & \mathit{Person}[\mathit{name}] \\
\mathit{Actor}[\mathit{name}] & \subseteq & \mathit{Person}[\mathit{name}]
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{Movie}[\mathit{director\_name}] & \subseteq & \mathit{Director}[\mathit{name}] \\
\mathit{Acts\_in}[\mathit{name}] & \subseteq & \mathit{Actor}[\mathit{name}] \\
\mathit{Acts\_in}[\mathit{title}] & \subseteq & \mathit{Movie}[\mathit{title}] \\
\mathit{Has\_Award}[\mathit{name}] & \subseteq & \mathit{Actor}[\mathit{name}]
\end{array}
$$

$$\begin{array}{lcl} \mathit{Has\_Award}[\mathit{prize}, \mathit{year}] & \subseteq & \mathit{Award}[\mathit{prize}, \mathit{year}] \\ \\ \mathit{Pariscope}[\mathit{theater\_name}] & \subseteq & \mathit{Theater}[\mathit{theater\_name}] \\ \mathit{Pariscope}[\mathit{title}] & \subseteq & \mathit{Movie}[\mathit{title}] \end{array}$$

The first group of ind's follows from ISA relationships; the second from restrictions on attribute ranges; and the third from restrictions on the components of complex values. All but one of the ind's here are unary, because all of the keys, except the key for *Award*, are based on a single attribute.

Preservation of Data

Suppose that $\mathbf{S}$ is a GSM schema with keys for primary classes, and $(\mathbf{R}, \Sigma \cup \Gamma)$ is a relational schema that simulates it, constructed in the fashion illustrated in Example 11.1.2, where Σ is the set of fd's and Γ is the set of ind's. As noted in criterion (1) at the beginning of this chapter, it is desirable that there be a natural one-to-one mapping τ from instances of $\mathbf{S}$ to instances of $(\mathbf{R}, \Sigma \cup \Gamma)$. To formalize this, two obstacles need to be overcome. First, we have not developed a query language for the GSM. (In fact, no query language has become widely accepted for any of the semantic data models. In contrast, some query languages for object-oriented database models are now gaining wide acceptance.) We shall overcome this obstacle by developing a rather abstract notion of "natural" for this context.

The second obstacle stems from the fact that OID-equivalent GSM instances hold essentially the same information. Thus we would expect OID-equivalent instances to map to the same relational instance.[1] To refine criterion (1) for this context, we are searching for a one-to-one mapping from $\mathit{Inst}(\mathbf{S})/\equiv_{OID}$ into $\mathit{Inst}(\mathbf{R}, \Sigma \cup \Gamma)$.

A mapping $\tau : \mathit{Inst}(\mathbf{S}) \to \mathit{Inst}(\mathbf{R}, \Sigma \cup \Gamma)$ is *OID consistent* if $\mathbf{I} \equiv_{OID} \mathbf{I}'$ implies $\tau(\mathbf{I}) = \tau(\mathbf{I}')$. In this case, we can view τ as a mapping with domain $\mathit{Inst}(\mathbf{S})/\equiv_{OID}$. The mapping τ *preserves the active domain* if for each $\mathbf{I} \in \mathit{Inst}(\mathbf{S})$, $\mathit{adom}(\tau(\mathbf{I})) = \mathit{adom}(\mathbf{I})$. [The *active domain* of a GSM instance $\mathbf{I}$, denoted $\mathit{adom}(\mathbf{I})$, is the set of all printables that occur in $\mathbf{I}$.]

The following can be verified (see Exercise 11.3):

Theorem 11.1.3 (Informal) Let $\mathbf{S}$ be a GSM schema with keys for primary classes, and let $(\mathbf{R}, \Sigma \cup \Gamma)$ be a relational simulation of $\mathbf{S}$. Then there is a function $\tau : \mathit{Inst}(\mathbf{S}) \to \mathit{Inst}(\mathbf{R}, \Sigma \cup \Gamma)$ such that τ is OID consistent and preserves the active domain, and such that $\tau : \mathit{Inst}(\mathbf{S})/\equiv_{OID} \to \mathit{Inst}(\mathbf{R}, \Sigma \cup \Gamma)$ is one-to-one and onto.

Properties of the Relational Schema

We now consider criteria (2) and (3) to highlight desirable properties of relational schemas that simulate GSM schemas.

[1] When artificial surrogates are used to represent OIDs in the relational database, one might have to use a notion of an "equivalent" relational database instances as well.

Criterion (2) for schema transformations concerns desirable properties of the target schema. We now describe three such properties resulting from the transformation of GSM schemas into relational ones.

Suppose again that $\mathbf{S}$ is a GSM schema with keys, and $(\mathbf{R}, \Sigma \cup \Gamma)$ is a relational simulation of it. We assume as before that no constraints hold for $\mathbf{S}$, aside from those implied by the constructs in $\mathbf{S}$ and the keys.

The three properties are as follows:

1. First, Σ is equivalent to a family of key dependencies; in the terminology of the next section, this means that each of the relation schemas obtained is in Boyce-Codd Normal Form (BCNF). Furthermore, the only mvd's satisfied by relations in $\mathbf{R}$ are implied by Σ, and so the relation schemas are in fourth normal form (4NF).
2. Second, the family Γ of ind's is acyclic (see Chapter 9). That is, there is no sequence $R_1[X_1] \subseteq R_2[Y_1]$, $R_2[X_2] \subseteq R_3[Y_2]$, $\ldots$, $R_n[X_n] \subseteq R_1[Y_n]$ of ind's in the set. By Theorem 9.4.5, this implies that logical implication can be decided for $(\Sigma \cup \Gamma)$ and that finite and unrestricted implication coincide.
3. Finally, each ind $R[X] \subseteq S[Y]$ in Γ *key based*. That is, Y is a (minimal) key of S under Σ.

Together these properties present a number of desirable features. In particular, dependency implication is easy to check. Given a database schema $\mathbf{R}$ and sets Σ of fd's and Γ of ind's over $\mathbf{R}$, Σ and Γ are *independent* if (1) for each fd σ over $\mathbf{R}$, $(\Sigma \cup \Gamma) \models \sigma$ implies $\Sigma \models \sigma$, and (2) for each ind γ over $\mathbf{R}$, $(\Sigma \cup \Gamma) \models \gamma$ implies $\Gamma \models \gamma$. Suppose that $\mathbf{S}$ is a GSM schema and that $(\mathbf{R}, \Sigma \cup \Gamma)$ is a relational simulation of $\mathbf{S}$. It can be shown that the three aforementioned properties imply that Σ and Γ are independent (see Exercise 11.4).

To conclude this section, we consider criterion (3). This criterion concerns the preservation of meta-data. We do not attempt to formalize this criterion for this context, but it should be clear that there is a close correspondence between the dependencies in $\Sigma \cup \Gamma$ and the constructs used in $\mathbf{S}$. In other words, the semantics of the application as expressed by $\mathbf{S}$ is also captured, in the relational representation, by the dependencies $\Sigma \cup \Gamma$.

The preceding discussion assumes that no dependency holds for $\mathbf{S}$, aside from those implied by the keys and the constructs in $\mathbf{S}$. However, in many cases constraints will be incorporated into $\mathbf{S}$ that are not directly implied by the structure of $\mathbf{S}$. For instance, recall Example 11.1.2, and suppose that the fd *Pariscope* : *theater_name*, *time* $\rightarrow$ *price* is true for the underlying data. The relational simulation will have to include this dependency and, as a result, the resulting relational schema may be missing some of the desirable features (e.g., the family of fd's is not equivalent to a set of keys and the schema is no longer in BCNF). This suggests that a semantic model might be used to obtain a coarse relational schema, which might be refined further using the techniques for improving relational schemas developed in the next section.

11.2 Normal Forms

In this section, we consider schema design based on the refinement of relational schemas and normal forms, which provide the basis for this approach. The articulation of these normal forms is arguably the main contribution of relational database theory to the realm of schema design. We begin the discussion by presenting two of the most prominent normal forms and a design strategy based on "decomposition." We then develop another normal form that overcomes certain technical problems of the first two, and describe an associated design strategy based on "synthesis." We conclude with brief comments on the relationship of ind's with decomposition.

When all the dependencies in a relational schema $(\mathbf{R}, \Sigma)$ are considered to be tagged, one can view the *database schema* as a set $\{(R_1, \Sigma_1), \ldots, (R_n, \Sigma_n)\}$, where each (R_j, Σ_j) is a relation schema and the R_j's are distinct. In particular, an *fd schema* is a relation schema (R, Σ) or database schema $(\mathbf{R}, \Sigma)$, where Σ is a set of tagged fd's; this is extended in the natural fashion to other classes of dependencies. Much of the work on refinement of relational schemas has focused on fd schemas and (fd + mvd) schemas. This is what we consider here. (The impact of the ind's is briefly considered at the end of this section.)

A normal form restricts the set of dependencies that are allowed to hold in a relation schema. The main purpose of the normal forms is to eliminate at least some of the redundancies and update anomalies that might otherwise arise. Intuitively, schemas in normal form are "good" schemas.

We introduce next two kinds of normal forms, namely BCNF and 4NF. (We will consider a third one, 3NF, later.) We then consider techniques to transform a schema into such desirable normal forms.

BCNF: Do Not Represent the Same Fact Twice

Recall the schema $(Movies[T(itle), D(irector), A(actor)], \{T \rightarrow D\})$ from Section 8.1. As discussed there, the *Movies* relation suffers from various anomalies, primarily because there is only one *Director* associated with each *Title* but possibly several *Actors.* Suppose that $(R[U], \Sigma)$ is a relation schema, $\Sigma \models X \rightarrow Y$, $Y \not\subseteq X$ and $\Sigma \not\models X \rightarrow U$. It is not hard to see that anomalies analogous to those of *Movies* can arise in R. Boyce-Codd normal form prohibits this kind of situation.

Definition 11.2.1 A relation schema $(R[U], \Sigma)$ is in *Boyce-Codd normal form* (BCNF) if $\Sigma \models X \rightarrow U$ whenever $\Sigma \models X \rightarrow Y$ for some $Y \not\subseteq X$. An fd schema $(\mathbf{R}, \Sigma)$ is in BCNF if each of its relation schemas is.

BCNF is most often discussed in cases where Σ involves only functional dependencies. In such cases, if (R, Σ) is in BCNF, the anomalies of Section 8.1 do not arise. An essential intuition underlying BCNF is, "Do not represent the same fact twice."

The question now arises: What does one do with a relation schema (R, Σ) that is not in BCNF? In many cases, it is possible to decompose this schema into subschemas $(R_1, \Sigma_1), \ldots, (R_n, \Sigma_n)$ without information loss. As a simple example, *Movies* can be decomposed into

$$\left\{ \begin{array}{l} (Movie\_director[TD], \{T \rightarrow D\}), \\ (Movie\_actors[TA], \emptyset) \end{array} \right\}$$

A general framework for decomposition is presented shortly.

4NF: Do Not Store Unrelated Information in the Same Relation

Consider the relation schema ($Studios[N(ame), D(irector), L(ocation)]$, $\{N \twoheadrightarrow D|L\}$). A tuple $\langle n, d, l \rangle$ is in *Studios* if director d is employed by the studio with name n and if this studio has an office in location l. Only trivial fd's are satisfied by all instances of this schema, and so it is in BCNF. However, update anomalies can still arise, essentially because the D and L values are independent from each other. This gives rise to the following generalization of BCNF[2]:

Definition 11.2.2 A relation schema $(R[U], \Sigma)$ is in *fourth normal form* (*4NF*) if

(a) whenever $\Sigma \models X \rightarrow Y$ and $Y \not\subseteq X$, then $\Sigma \models X \rightarrow U$

(b) whenever $\Sigma \models X \twoheadrightarrow Y$ and $Y \not\subseteq X$, then $\Sigma \models X \rightarrow U$.

An (fd + mvd) schema $(\mathbf{R}, \Sigma)$ is in 4NF if each of its relation schemas is.

It is clear that if a relation schema is in 4NF, then it is in BCNF. It is easily seen that *Studios* can be decomposed into two 4NF relations, without loss of information and that the resulting relation schemas do not have the update anomalies mentioned earlier. An essential intuition underlying 4NF is, "Do not store unrelated information in the same relation."

The General Framework of Decomposition

One approach to refining relational schemas is *decomposition*. In this approach, it is usually assumed that the original schema consists of a single wide relation containing all attributes of interest. This is referred to as the *pure universal relation assumption,* or *pure URA*. A relaxation of the pure URA, called the "weak URA," is considered briefly in Section 11.3.

The pure URA is a simplifying assumption, because in practice the original schema is likely to consist of several tables, each with its own dependencies. In that case, the design process described for the pure URA is applied separately to each table. We adopt the pure URA here. In this context, the schema transformation produced by the design process consists of decomposing the original table into smaller tables by using the projection operator. (In an alternative approach, selection is used to yield so-called *horizontal decompositions*.)

We now establish the basic framework of decompositions. Let $(U[Z], \Sigma)$ be a relation schema. A *decomposition* of $(U[Z], \Sigma)$ is a database schema $\mathbf{R} = \{R_1[X_1], \ldots, R_n[X_n]\}$ with dependencies Γ, where $\cup\{X_j \mid j \in [1, n]\} = Z$. (The relation name '$U$' is used to suggest that it is a "universal" relation.) In the sequel, we often use relation names U (R_i) and attribute sets Z (X_i), interchangeably if ambiguity does not arise.

[2] The motivation behind the names of several of the normal forms is largely historical; see the Bibliographic Notes.

We now consider the three criteria for schema transformation in the context of decomposition. As already suggested, criterion (2) is evaluated in terms of the normal forms. With regard to the preservation of data (1), the "natural" mapping from R to $\mathbf{R}$ is obtained by projection: The *decomposition mapping* of $\mathbf{R}$ is the function $\pi_{\mathbf{R}} : Inst(U) \to Inst(\mathbf{R})$ such that for $I \in inst(U)$, we have $\pi_{\mathbf{R}}(I)(R_j) = \pi_{R_j}(I)$. Criterion (1) says that the decomposition should not lose information when I is replaced by its projections (i.e., it should be one-to-one).

A natural property implying that a decomposition is one-to-one is that the original instance can be obtained by joining the component relations. Formally, a decomposition is said to have the *lossless join* property if for each instance $\mathbf{I}$ of (U, Σ) the join of the projections is the original instance, i.e., $\bowtie(\pi_{\mathbf{R}}(\mathbf{I})) = \mathbf{I}$. It is easy to test if a decomposition $\mathbf{R} = \{R_1, \ldots, R_n\}$ of (U, Σ) has the lossless join property. Consider the query $q(I) = \pi_{R_1}(I) \bowtie \cdots \bowtie \pi_{R_n}(I)$. The lossless join property means that $q(I) = I$ for every instance I over (U, Σ). But $q(I) = I$ simply says that I satisfies the jd $\bowtie[\mathbf{R}]$. Thus we have the following:

Theorem 11.2.3 Let (U, Σ) be a (full dependencies) schema and $\mathbf{R}$ a decomposition for (U, Σ). Then $\mathbf{R}$ has the lossless join property iff $\Sigma \models \bowtie[\mathbf{R}]$.

The preceding implication can be tested using the chase (see Chapter 8), as illustrated next.

Example 11.2.4 Recall the schema $(Movies[TDA], \{T \to D\})$. As suggested earlier, a decomposition into BCNF is $\mathbf{R} = \{TD, TA\}$. This decomposition has the lossless join property. The tableau associated with the jd $\sigma = \bowtie[TD, TA]$ is as follows:

	T	D	A
T_σ	t	d	a_1
	t	d_1	a
t_σ	t	d	a

Consider the chase of $\langle T_\sigma, t_\sigma \rangle$ with $\{T \to D\}$. Because the two first tuples agree on the T column, d and d_1 are merged because of the fd. Thus $\langle t, d, a \rangle \in chase(T_\sigma, t_\sigma, \{T \to D\})$. Hence $T \to D$ implies the jd σ, so $\mathbf{R}$ has the lossless join property. (See also Exercise 11.9.)

Referring to the preceding example, note that it is possible to represent information in $\mathbf{R}$ that cannot be directly represented in *Movies*. Specifically, in the decomposed schema we can represent a movie with a director but no actors and a movie with an actor but no director. This indicates, intuitively, that a decomposed schema may have more information capacity

than the original (see Exercise 11.23). In practice, this additional capacity is exploited; in fact, it provides part of the solution of so-called deletion anomalies.

Remark 11.2.5 In the preceding example, we used the natural join operator to reconstruct decompositions. Interestingly, there are cases in which the natural join does not suffice. To show that a decomposition is one-to-one, it suffices to exhibit an inverse to the projection, called a *reconstruction mapping*. If Σ is permitted to include very general constraints expressed in first-order logic that may not be dependencies per se, then there are one-to-one decompositions whose reconstruction mappings are *not* the natural join (see Exercise 11.20).

We now consider criterion (3), the preservation of meta-data. In the context of decomposition, this is formalized in terms of "dependency preservation": Given schema (U, Σ), which is replaced by a decomposition $\mathbf{R} = \{R_1, \ldots, R_n\}$, we would like to find for each j a family Γ_j of dependencies over R_j such that $\cup_j \Gamma_j$ is equivalent to the original Σ. In the case where Σ is a set of fd's, we can make this much more precise. For $V \subseteq U$, let

$$\pi_V(\Sigma) = \{X \to A \mid XA \subseteq V \text{ and } \Sigma \models X \to A\},$$

let $\Gamma_j = \pi_{X_j}(\Sigma)$, and let $\Gamma = \cup_j \Gamma_j$. Obviously, $\Sigma \models \Gamma$. (See Proposition 10.2.4.) Intuitively, Γ consists of the dependencies in Σ^* that are local to the relations in the decomposition $\mathbf{R}$. The decomposition $\mathbf{R}$ is said to be *dependency preserving* iff $\Gamma \equiv \Sigma$. In other words, Σ can be enforced by the dependencies local in the decomposition. It is easy to see that the decomposition of Example 11.2.4 is dependency preserving.

Given an fd schema (U, Σ) and $V \subseteq U$, $\pi_V(\Sigma)$ has size exponential in V, simply because of trivial fd's. But perhaps there is a smaller set of fd's that is equivalent to $\pi_V(\Sigma)$. A *cover* of a set Γ of fd's is a set Γ' of fd's such that $\Gamma' \equiv \Gamma$. Unfortunately, in some cases the smallest cover for a projection $\pi_V(\Sigma)$ is exponential in the size of Σ (see Exercise 11.11).

What about projections of sets of mvd's? Suppose that Σ is a set of fd's and mvd's over U. Let $V \subseteq U$ and

$$\pi_V^{mvd}(\Sigma) = \{[X \twoheadrightarrow (Y \cap V) | (Z \cap V)] \mid [X \twoheadrightarrow Y | Z] \in \Sigma^* \text{ and } X \subseteq V\}.$$

Consider a decomposition $\mathbf{R}$ of (U, Σ). Viewed as constraints on U, the sets $\pi_{R_j}^{mvd}(\Sigma)$ are now embedded mvd's. As we saw in Chapter 10, testing implication for embedded mvd's is undecidable. However, the issue of testing for dependency preservation in the context of decompositions involving fd's and mvd's is rather specialized and remains open.

Fd's and Decomposition into BCNF

We now present a simple algorithm for decomposing an fd schema (U, Σ) into BCNF relations. The decomposition produced by the algorithm has the lossless join property but is not guaranteed to be dependency preserving.

We begin with a simple example.

EXAMPLE 11.2.6 Consider the schema (U, Σ), where U has attributes

TITLE *D_NAME* *TIME* *PRICE*
TH_NAME *ADDRESS* *PHONE*

and Σ contains

$FD1$: $TH\_NAME \rightarrow ADDRESS, PHONE$
$FD2$: $TH\_NAME, TIME, TITLE \rightarrow PRICE$
$FD3$: $TITLE \rightarrow D\_NAME$

Intuitively, schema (U, Σ) represents a fragment of the real-world situation represented by the semantic schema **CINEMA-SEM**.

A first step toward transforming this into a BCNF schema is to decompose using $FD1$, to obtain the database schema

$$\left\{ \begin{array}{l} (\{TH\_NAME, ADDRESS, PHONE\}, \{FD1\}), \\ (\{TH\_NAME, TITLE, TIME, PRICE, D\_NAME\}, \{FD2, FD3\}) \end{array} \right\}$$

Next $FD3$ can be used to split the second relation, obtaining

$$\left\{ \begin{array}{l} (\{TH\_NAME, ADDRESS, PHONE\}, \{FD1\}) \\ (\{TITLE, D\_NAME\}, \{FD3\}) \\ (\{TH\_NAME, TITLE, TIME, PRICE\}, \{FD2\}) \end{array} \right\}$$

which is in BCNF. It is easy to see that this decomposition has the lossless join property and is dependency preserving. In fact, in this case, we obtain the same relational schema as would result from starting with a semantic schema.

We now present the following:

ALGORITHM 11.2.7 (BCNF Decomposition)

Input: A relation schema (U, Σ), where Σ is a set of fd's.

Output: A database schema $(\mathbf{R}, \Gamma)$ in BCNF

1. Set $(\mathbf{R}, \Gamma) := \{(U, \Sigma)\}$.
2. Repeat until $(\mathbf{R}, \Gamma)$ is in BCNF:
   (a) Choose a relation schema $(S[V], \Omega) \in \mathbf{R}$ that is not in BCNF.
   (b) Choose nonempty, disjoint $X, Y, Z \subset V$ such that
      (i) $XYZ = V$;
      (ii) $\Omega \models X \rightarrow Y$; and
      (iii) $\Omega \not\models X \rightarrow A$ for each $A \in Z$.
   (c) Replace $(S[V], \Omega)$ in $\mathbf{R}$ by $(S_1[XY], \pi_{XY}(\Omega))$ and $(S_2[XZ], \pi_{XZ}(\Omega))$.
   (d) If there are $(S[V], \Omega)$, $(S'[V'], \Omega')$ in $\mathbf{R}$ with $V \subseteq V'$, then remove $S([V], \Omega)$ from $\mathbf{R}$.

It is easily seen that the preceding algorithm terminates [each iteration of the loop eliminates at least one violation of BCNF among finitely many possible ones]. The following is easily verified (see Exercise 11.10):

THEOREM 11.2.8 The BCNF Decomposition Algorithm yields a BCNF schema and a decomposition that has the lossless join property.

What is the complexity of running the BCNF Decomposition Algorithm? The main expenses are (1) examining subschemas $(S[V], \Omega)$ to see if they are in BCNF and, if not, finding a way to decompose them; and (2) computing the projections of Ω. (1) is polynomial, but (2) is inherently exponential (see Exercise 11.11). This suggests a modification to the algorithm, in which only the relational schemas $S[V]$ are computed at each stage, but $\Omega = \pi_V(\Sigma)$ is not. However, the problem of determining, given fd schema (U, Σ) and $V \subseteq U$, whether $(V, \pi_V(\Sigma))$ is in BCNF is co-NP-complete (see Exercise 11.12). Interestingly, a polynomial time algorithm does exist for finding *some* BCNF decomposition of an input schema (U, Σ) (see Exercise 11.13).

When applying BCNF decomposition to the schema of Example 11.2.6, the same result is achieved regardless of the order in which the dependencies are applied. This is not always the case, as illustrated next.

EXAMPLE 11.2.9 Consider $(ABC, \{A \to B, B \to C\})$. This has two BCNF decompositions

$$\mathbf{R}_1 = \{(AB, \{A \to B\}), (BC, \{B \to C\})\}$$
$$\mathbf{R}_2 = \{(AB, \{A \to B\}), (AC, \emptyset)\}.$$

Note that $\mathbf{R}_1$ is dependency preserving, but $\mathbf{R}_2$ is not.

Fd's, Dependency Preservation, and 3NF

It is easy to check that the schemas in Examples 11.2.4, 11.2.6, and 11.2.9 have dependency-preserving decompositions into BCNF. However, this is not always achievable, as shown by the following example.

EXAMPLE 11.2.10 Consider a schema *Lectures*[*C*(*ourse*), *P*(*rofessor*), *H*(*our*)], where tuple $\langle c, p, h\rangle$ indicates that course c is taught by professor p at hour h. We assume that *Hour* ranges over weekday-time pairs (e.g., Tuesday at 4PM) and that a given course may have lectures during several hours each week. Assume that the following two dependencies are to hold:

$$\Sigma = \left\{ \begin{array}{c} C \to P \\ PH \to C \end{array} \right\}.$$

In other words, each course is taught by only one professor, and a professor can teach only one course at a given hour.

The schema (*Lectures*, Σ) is not in BCNF because $\Sigma \models C \to P$, but $\Sigma \not\models C \to H$. Applying the BCNF Decomposition Algorithm yields $\mathbf{R} = \{(CP, \{C \to P\}), (CH, \emptyset)\}$.

It is easily seen that $\{CP : C \to P\} \not\models \Sigma$, and so this decomposition does not preserve dependencies. A simple case analysis shows that there is no BCNF decomposition of *Lectures* that preserves dependencies.

This raises the question: Is there a less restrictive normal form for fd's so that a lossless join decomposition that preserves dependencies can always be found? The affirmative answer is based on "third normal form" (3NF). To define it, we need some auxiliary notions. Suppose that $(R[U], \Sigma)$ is an fd schema. A *superkey* of R is a set $X \subseteq U$ such that $\Sigma \models X \to U$. A *key* of R is a minimal superkey. A *key attribute* is an attribute $A \in U$ that is in some key of R. We now have the following:

Definition 11.2.11 An fd schema (U, Σ) is in *third normal form* (*3NF*) if whenever $X \to A$ is a nontrivial fd implied by Σ, then either X is a superkey or A is a key attribute. An fd schema $(\mathbf{R}, \Sigma)$ is in 3NF if each of its components is.

Example 11.2.12 Recall the schema (*Lectures*,$\{C \to P, PH \to C\}$) described in Example 11.2.10. Here PH is a key, so P is a key attribute. Thus the schema is in 3NF.

A 3NF Decomposition Algorithm can be defined in analogy to the BCNF Decomposition Algorithm. We present an alternative approach, generally referred to as "synthesis."

Given a set Σ of fd's, a *minimal cover* of Σ is a set Σ' of fd's such that

(a) each dependency in Σ' has the form $X \to A$, where A is an attribute;

(b) $\Sigma' \equiv \Sigma$;

(c) no proper subset of Σ' implies Σ; and

(d) for each dependency $X \to A$ in Σ', there is no $Y \subset X$ such that $\Sigma \models Y \to A$.

A minimal cover can be viewed as a reduced representative for a set of fd's. It is straightforward to develop a polynomial time algorithm for producing a minimal cover of a set of fd's (see Exercise 11.16).

We now have the following:

Algorithm 11.2.13 (3NF Synthesis)

Input: A relation schema (U, Σ), where Σ is a set of fd's that is a minimal cover. We assume that each attribute of U occurs in at least one fd of Σ.

Output: An fd schema $(\mathbf{R}, \Gamma)$ in 3NF

1. If there is an fd $X \to A$ in Σ, where $XA = U$, then output (U, Σ).
2. Otherwise
   (a) for each fd $X \to A$ in Σ, include the relational schema $(XA, \{X \to A\})$ in the output schema $(\mathbf{R}, \Gamma)$; and

(b) choose a key X of U under Σ, and include $(X, \emptyset)$ in the output.

A central aspect of this algorithm is to form a relation XA for each fd $X \to A$ in Σ. Intuitively, then, the output relations result from combining or "synthesizing" attributes rather than decomposing the full attribute set.

The following is easily verified (see Exercise 11.17):

THEOREM 11.2.14 The 3NF Synthesis Algorithm decomposes a relation schema into a database schema in 3NF that has the lossless join property and preserves dependencies.

Several improvements to the basic 3NF Synthesis Algorithm can be made easily. For example, different schemas obtained in step (2.a) can be merged if they come from fd's with the same left-hand side. Step (2.b) is not needed if step (2.a) already produced a schema whose set of attributes is a superkey for (U, Σ). In many practical situations, it may be appropriate to omit step (2.b) of the algorithm. In that case, the decomposition preserves dependencies but does not necessarily satisfy the lossless join property.

In the preceding algorithm, it was assumed that each attribute of U occurs in at least one fd of Σ. Obviously, this may not always be the case, for example, the attribute *A_NAME* in Example 11.2.15b does not participate in fd's. One approach to remedy this situation is to introduce symbolic fd's. For instance, in that example one might include the fd *TITLE, A_NAME* $\to \omega_1$, where ω_1 is a new attribute. One relation produced by the algorithm will be $\{TITLE, A\_NAME, \omega_1\}$. As a last step, attributes such as ω_1 are removed.

In Example 11.2.9 we saw that the output of a BCNF decomposition may depend on the order in which fd's are applied. In the case of the preceding algorithm for 3NF, the minimal cover chosen greatly impacts the final result.

Mvd's and Decomposition into 4NF

A fundamental problem with BCNF decomposition and 3NF synthesis as just presented is that they do not take into account the impact of mvd's.

EXAMPLE 11.2.15 (a) The schema (*Studios*[*N*(*ame*), *D*(*irector*), *L*(*ocation*)], $\{N \twoheadrightarrow D|L\}$) is in BCNF and 3NF but has update anomalies. The mvd suggests a decomposition into ($\{Name, Director\}$, $\{Name, Location\}$).

(b) A related issue is that BCNF decompositions may not separate attributes that intuitively should be separated. For example, consider again the schema of Example 11.2.6, but suppose that the attribute *A_NAME* is included to denote actor names. Following the same decomposition steps as before, we obtain the schema

$$\left\{ \begin{array}{l} (\{TH\_NAME, ADDRESS, PHONE\}, \{FD1\}), \\ \\ (\{TITLE, D\_NAME\}, \{FD3\}), \\ \\ (\{TH\_NAME, TITLE, TIME, PRICE, A\_NAME\}, \{FD2\}) \end{array} \right\}$$

which can be further decomposed to

$$\left\{\begin{array}{l}(\{TH\_NAME, ADDRESS, PHONE\}, \{FD1\}),\\ (\{TITLE, D\_NAME\}, \{FD3\}),\\ (\{TH\_NAME, TITLE, TIME, PRICE\}, \{FD2\}),\\ (\{TH\_NAME, TITLE, TIME, A\_NAME\}, \emptyset)\end{array}\right\}$$

Although there is a connection in the underlying data between *TITLE* and *A_NAME*, the last relation here is unnatural. If we assume that the mvd $TITLE \twoheadrightarrow A\_NAME$ is incorporated into the original schema, we can further decompose the last relation and apply a step analogous to (2d) of the BCNF Decomposition Algorithm to obtain

$$\left\{\begin{array}{l}(\{TH\_NAME, ADDRESS, PHONE\}, \{FD1\}),\\ (\{TITLE, D\_NAME\}, \{FD3\}),\\ (\{TH\_NAME, TITLE, TIME, PRICE\}, \{FD2\}),\\ (\{TITLE, A\_NAME\}, \emptyset)\end{array}\right\}$$

Fourth normal form (4NF) was originally developed to address these kinds of situations. As suggested by the preceding example, an algorithm yielding 4NF decompositions can be developed along the lines of the BCNF Decomposition Algorithm. As with BCNF, the output of 4NF decomposition is a lossless join decomposition that is not necessarily dependency preserving.

A Note on Ind's

In relational schema design starting with a semantic data model, numerous ind's are typically generated. In contrast, the decomposition and synthesis approaches for refining relational schemas as presented earlier do not take ind's into account. It is possible to incorporate ind's into these approaches, but the specific choice of ind's is dependent on the intended semantics of the target schema.

Example 11.2.16 Recall the schema $(Movies[TDA], \{T \rightarrow D\})$ and decomposition into $(R_1[TD], \{T \rightarrow D\})$ and $(R_2[TA], \emptyset)$.

(a) If all movies must have a director and at least one actor, then both $R_1[T] \subseteq R_2[T]$ and $R_2[T] \subseteq R_1[T]$ should be included. In this case, the mapping from *Movies* to its decomposed representation is one-to-one and onto.

(b) If the fd $T \rightarrow D$ is understood to mean that there is a *total* function from movies to directors, but movies without actors are permitted, then the ind $R_2[T] \subseteq R_1[T]$ should be included.

(c) Finally, suppose the fd $T \rightarrow D$ is understood to mean that each movie has at most one director (i.e., it is a partial function), and suppose that a movie can have no actor. Then an additional relation $R_3[T]$ should be added to hold the titles of all movies, along with ind's $R_1[T] \subseteq R_3[T]$ and $R_2[T] \subseteq R_3[T]$.

More generally, what if one is to refine a relational schema $(\mathbf{R}, \Sigma \cup \Gamma)$, where Σ is a set of tagged fd's and mvd's and Γ is a set of ind's? It may occur that there is an ind $R_i[X] \subseteq R_j[Y]$, and either X or Y is to be "split" as the result of a decomposition step. The desired semantics of the target schema can be used to select between a variety of heuristic approaches to preserving the semantics of this ind. If Γ consists of unary ind's, such splitting cannot occur. Speaking intuitively, if the ind's of Γ are key based, then the chances of such splitting are reduced.

11.3 Universal Relation Assumption

In the preceding section, we saw that the decomposition and synthesis approaches to relational schema design assume the pure URA. This section begins by articulating some of the implications that underly the pure URA. It then presents the "weak URA," which provides an intuitively natural mechanism for viewing a relational database instance **I** as if it were a universal relation.

Underlying Assumptions

Suppose that an fd schema $(U[Z], \Sigma)$ is given and that decomposition or synthesis will be applied. One of several different database schemas might be produced, but presumably all of them carry roughly the same semantics. This suggests that the attributes in Z can be grouped into relation schemas in several different ways, without substantially affecting their underlying semantics. Intuitively, then, it is the attributes themselves (along with the dependencies in Σ), rather than the attributes as they occur in different relation schemas, that carry the bulk of the semantics in the schema. The notion that the attributes can represent a substantial portion of the semantics of an application is central to schema design based on the pure URA.

When decomposition and synthesis were first introduced, the underlying implications of this notion were not well understood. Several intuitive assumptions were articulated that attempted to capture these implications. We describe here two of the most important assumptions. Any approach to relational schema design based on the pure URA should also abide by these two assumptions.

Universal Relation Scheme Assumption: This states that if an attribute name appears in two or more places in a database schema, then it refers to the same entity set in each place. For example, an attribute name *Number* should not be used for both serial numbers and employee numbers; rather two distinct attribute names *Serial#* and *Employee#* should be used.

Unique Role Assumption: This states that for each set of attributes there is a unique rela-

tionship between them. This is sometimes weakened to say that there may be several relationships, but one is deemed primary. This is illustrated in the following example.

Example 11.3.1 (a) Recall in Example 11.2.15(b) that *D_NAME* and *A_NAME* were used for director and actor names, respectively. This is because there were two possible relationships between movies and persons.

(b) For a more complicated example, consider a schema for bank branches that includes attributes for *B*(*ranch*), *L*(*oan*), (*checking*) *A*(*ccount*), and *C*(*ustomer*). Suppose there are four relations

BL, which holds data about branches and loans they have given

BA, which holds data about branches and checking accounts they provide

CL, which holds data about customers and loans they have

CA, which holds data about customers and checking accounts they have.

This design does not satisfy the unique role assumption, mainly because of the cycle in the schema. For example, consider the relationship between branches and customers. In fact, there are two relationships—via loans and via accounts. Thus a request for "the" data in the relationship between banks and customers is somewhat ambiguous, because it could mean tuples stemming from either of the two relationships or from the intersection or union of both of them.

One solution to this ambiguity is to "break" the cycle. For example, we could replace the *Customer* attribute by the two attributes *L-C*(*ustomer*) and *A-C*(*ustomer*). Now the user can specify the desired relationship by using the appropriate attribute.

The Weak Universal Relation Assumption

Suppose that schema (U, Σ) has decomposition $(\mathbf{R}, \Gamma)$ (with $\mathbf{R} = \{R_1, \ldots, R_n\}$). When studying decomposition, we focused primarily on instances $\mathbf{I}$ of $(\mathbf{R}, \Gamma)$ that were the image of some instance I of (U, Σ) under the decomposition mapping $\pi_{\mathbf{R}}$. In particular, such instances $\mathbf{I}$ are *globally consistent*. [Recall from Chapter 6 that instance $\mathbf{I}$ is *globally consistent* if for each $j \in [1, n]$, $\pi_{R_j}(\bowtie \mathbf{I}) = \mathbf{I}(R_j)$; i.e., no tuple of $\mathbf{I}(R_j)$ is dangling relative to the full join.] However, in many practical situations it might be useful to use the decomposed schema $\mathbf{R}$ to store instances $\mathbf{I}$ that are not globally consistent.

Example 11.3.2 Recall the schema (*Movies*[*TDA*], $\{T \rightarrow D\}$) from Example 11.2.4 and its decomposition $\{TD, TA\}$. Suppose that for some movie the director is known, but no actors are known. As mentioned previously, this information is easily stored in the decomposed database, but not in the original. The impossibility of representing this information in the original schema was one of the anomalies that motivated the decomposition in the first place.

Suppose that fd schema (U, Σ) has decomposition $(\mathbf{R}, \Gamma) = \{(R_1, \Gamma_1), \ldots, (R_n, \Gamma_n)\}$. Suppose also that $\mathbf{I}$ is an instance of $\mathbf{R}$ such that (1) $\mathbf{I}(R_j) \models \Gamma_j$ for each j, but (2) $\mathbf{I}$ is

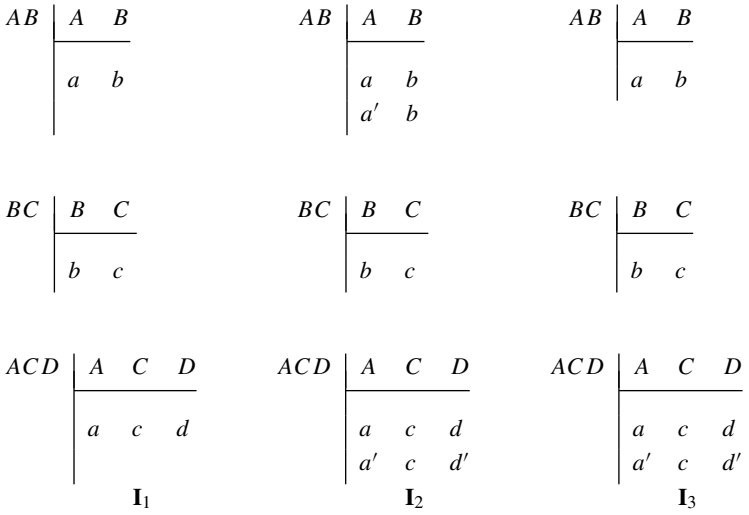

Figure 11.4: Instances illustrating weak URA

not necessarily globally consistent. Should $\mathbf{I}$ be considered a "valid" instance of schema $(\mathbf{R}, \Gamma)$? More generally, given a schema (U, Σ), a decomposition $\mathbf{R}$ of U, and a (not necessarily globally consistent) instance $\mathbf{I}$ over $\mathbf{R}$, how should we define the notion of "satisfaction" of Σ by $\mathbf{I}$?

The *weak universal relation assumption* (*weak URA*) provides one approach for answering this question. Under the weak URA, we say that $\mathbf{I}$ *satisfies* Σ if there is *some* instance $J \in sat(U, \Sigma)$ such that $\mathbf{I}(R_j) \subseteq \pi_{R_j}(J)$ for each $j \in [1, n]$. In this case, J is called a *weak instance* for $\mathbf{I}$.

Example 11.3.3 Let $U = \{ABCD\}$, $\Sigma = \{A \to B, BC \to D\}$, and $\mathbf{R} = \{AB, BC, ACD\}$. Consider the three instances of $\mathbf{R}$ shown in Fig. 11.4. The instance $\mathbf{I}_1$ satisfies Σ under the weak URA, because $J_1 = \{\langle a, b, c, d\rangle\}$ is a weak instance.

On the other hand, $\mathbf{I}_2$, which contains $\mathbf{I}_1$, does not satisfy Σ under the weak URA. To see this, suppose that J_2 is a weak instance for $\mathbf{I}_2$. Then J_2 must contain the following (not necessarily distinct) tuples:

$$t_1 = \langle a, b, c_1, d_1\rangle$$
$$t_2 = \langle a', b, c_2, d_2\rangle$$
$$t_3 = \langle a_3, b, c, d_3\rangle$$
$$t_4 = \langle a, b_4, c, d\rangle$$
$$t_5 = \langle a', b_5, c, d'\rangle$$

where the subscripted constants may be new. Because $J_2 \models A \to B$, by considering the

pairs $\langle t_1, t_4\rangle$ and $\langle t_2, t_5\rangle$, we see that $b_4 = b_5 = b$. Next, because $J_2 \models BC \rightarrow D$, and by considering the pair $\langle t_4, t_5\rangle$, we have that $d = d'$, a contradiction.

Finally, $\mathbf{I}_3$ does satisfy Σ under the weak URA.

As suggested by the preceding example, testing whether an instance $\mathbf{I}$ over $\mathbf{R}$ is a weak instance of (U, Σ) for a set of fd's Σ can be performed using the chase. To do that, it suffices to construct a table over U by padding the tuples from each R_j with distinct new variables. The resulting table is chased with the dependencies in Σ. If the chase fails, there is no weak instance for $\mathbf{I}$. On the other hand, a successful chase provides a weak instance for $\mathbf{I}$ by simply replacing each remaining variable with a distinct new constant.

This yields the following (see Exercise 11.27):

THEOREM 11.3.4 Let Σ be a set of fd's over U and $\mathbf{R}$ a decomposition of U. Testing whether $\mathbf{I}$ over $\mathbf{R}$ satisfies Σ under the weak URA can be performed in polynomial time.

Of course, the chasing technique can be extended to arbitrary egd's, although the complexity jumps to EXPTIME-complete.

What about full tgd's? Recall that full tgd's can always be satisfied by adding new tuples to an instance. Let Σ be a set of full dependencies. It is easy to see that $\mathbf{I}$ satisfies Σ under the weak URA iff $\mathbf{I}$ satisfies $\Sigma^* \cap \{\sigma \mid \sigma \text{ is an egd}\}$ under the weak URA.

Querying under the Weak URA

Let (U, Σ) be a schema, where Σ is a set of full dependencies, and let $\mathbf{R}$ be a decomposition of U. Let us assume the weak URA, and suppose that database instance $\mathbf{I}$ over $\mathbf{R}$ satisfies Σ. How should queries against $\mathbf{I}$ be answered? One approach is to consider the query against all weak instances for $\mathbf{I}$ and then take the intersection of the answers. That is,

$$q_{weak}(\mathbf{I}) = \cap\{q(I) \mid I \text{ is a weak instance of } \mathbf{I}\}.$$

We develop now a constructive method for computing q_{weak}.

Given instance $\mathbf{I}$ of $\mathbf{R}$, the *representative instance* of $\mathbf{I}$ is defined as follows: For each component I_j of $\mathbf{I}$, let I'_j be the result of extending I_j to be a free instance over U by padding tuples with distinct variables. Set $I' = \cup\{I'_j \mid j \in [1, n]\}$. Now apply the chase using Σ to obtain the representative instance $rep(\mathbf{I}, \Sigma)$ (or the empty instance, if two distinct constants are to be identified). Note that some elements of $rep(\mathbf{I}, \Sigma)$ may have variables occurring in them.

For $X \subseteq U$, let $\pi{\downarrow}_X(rep(\mathbf{I}, \Sigma))$ denote the set of tuples (i.e., with no variables present) in $\pi_X(rep(\mathbf{I}, \Sigma))$. The following can now be verified (see Exercise 11.28).

PROPOSITION 11.3.5 Let (U, Σ), $\mathbf{R}$ and $\mathbf{I}$ be as above, and let $X \subseteq U$. Then

(a) $[\pi_X]_{weak}(\mathbf{I}) = \pi{\downarrow}_X(rep(\mathbf{I}, \Sigma))$.

(b) If Σ is a set of fd's, then $[\pi_X]_{weak}(\mathbf{I})$ can be computed in PTIME.

This proposition provides the basis of a constructive method for evaluating an arbitrary algebra query q under the weak URA. Furthermore, if Σ is a set of fd's, then evaluating q will take time at most polynomial in the size of the input instance. This approach can be generalized to the case where Σ is a set of full dependencies but computing the projection is EXPTIME-complete.

Bibliographic Notes

The recent book [MR92] provides an in-depth coverage of relational schema design, including both the theoretical underpinnings and other, less formal factors that go into good design. Extensive treatments of the topic are also found in [Dat86, Fv89, Ull88, Vos91]. References [Ken78, Ken79, Ken89] illustrate the many difficulties that arise in schema design, primarily with a host of intriguing examples that show how skilled the human mind is at organizing diverse information and how woefully limiting data models are.

Surveys of semantic data models include [Bor85, HK87, PM88], and the book [TL82]; [Vos91] includes a chapter on this topic. Prominent early semantic data models include the Entity-Relationship (ER) model [Che76] (see also [BLN86, MR92, TYF86]), the Functional Data Model [Shi81, HK81], the Semantic Data Model [HM81], and the Semantic Binary Data Model [Abr74]. An early attempt to incorporate semantic data modeling constructs into the relational model is RM/T [Cod79]; more recently there have been various extensions of the relational model to incorporate object-oriented data modeling features (e.g., [SJGP90]). Many commercial systems support "tuple IDs," which can be viewed as a form of OID. Galileo [ACO85], Taxis [MBW80], and FQL [BFN82] are programming languages that support constructs stemming from semantic data models. The IFO [AH87] model is a relatively simple, formal semantic data model that subsumes the structural components of the aforementioned semantic models and several others. Reference [AH87] clarifies issues concerning ISA hierarchies in semantic schemas (see also [BLN86, Cod79, DH84] and studies the propagation of updates.

Reference [Che76] describes a translation of the ER model into the relational model, so that the resulting schema is in BCNF. From a practical perspective, this has become widely accepted as *the* method of choice for designing relational schemas; [TYF86] provides a subsequent perspective on this approach. There has also been considerable work on understanding the properties of relational schemas resulting from ER schemas and mapping relational schemas into ER ones. Reference [MR92] provides an in-depth discussion of this area.

Reference [LV87] presents a translation from a semantic to the relational model and studies the constraints implied for the relational schema, including cardinality constraints. The logical implication of constraints within a semantic model schema is studied in [CL94]. References [Lie80, Lie82] study the relationship of schemas from the network and relational models.

At a fundamental level, an important aspect of schema design is to replace one schema with another that can hold essentially the same information. This raises the issue of developing formal methods for comparing the *relative information capacity* of different schemas. Early work in this direction for the relational model includes [AABM82] and [BMSU81] (see Exercise 11.22). More abstract work is found in [HY84, Hul86] (see Exercises 11.23 and 11.24), which forms the basis for Theorem 11.1.3. Reference [MS92]

provides justification for translations from the Entity-Relationship model into the relational model using notions of relative information capacity. Formal notions of relative information capacity have also been applied in the context of schema integration and translation [MIR93] and heterogeneous databases [MIR94]. A very abstract framework for comparing schemas from different data models is proposed in [AT93].

The area of normal forms and relational database design was studied intensively in the 1970s and early 1980s. Much more complete coverage of this topic than presented here may be found in [Dat86, Mai83, Ull88, Vos91]. We mention some of the most important papers in this area. First normal form [Cod70] is actually fundamental to the relational model: A relation is in *first normal form* (1NF) if each column contains atomic values. In Chapter 20 this restriction shall be relaxed to permit relations some of whose columns themselves hold relations (which again may not be in first normal form). References [Cod71, Cod72a] raised the issue of update anomalies and initiated the search for normal forms that prevent them by introducing second and third normal forms. The definition of 3NF used here is from [Zan82]. (Second normal form is less restrictive than third normal form.) Boyce-Codd normal form (BCNF) was introduced in [Cod74] to provide a normal form simpler than 3NF. Another improvement of 3NF is proposed in [LTK81]. Fourth normal form was introduced in [Fag77b]; Example 11.2.15 is inspired from that reference. Even richer normal forms include project-join normal form (PJ/NF) [Fag79] and domain-key normal form [Fag81].

In addition to introducing second and third normal form, [Cod72a] initiated the search for normalization algorithms by proposing the first decomposition algorithms. This spawned other research on decomposition [DC72, RD75, PJ81] and synthesis [BST75, Ber76b, WW75]. The fact that these two criteria are not equivalent was stressed in [Ris77], where it is proposed that both be attempted. Early surveys on these approaches to relational design include [BBG78, Fag77a, Ris78]. Algorithms for synthesis into 3NF include [Ber76b, BDB79], for decomposition into BCNF include [TF82], and for decomposition into 4NF include [Fag77b]. Computational issues raised by decompositions are studied in [LO78, BB79, FJT83, TF82] and elsewhere. Reference [Got87] presents a good heuristic for finding covers of the projection of a set of fd's. The 3NF Synthesis Algorithm presented in this chapter begins with a *minimal cover* of a set of fd's; [Mai80] shows that minimal covers can be found in polynomial time.

The more formal study of decompositions and their properties was initiated in [Ris77], which considered decompositions into two-element sets and proposed the notion of independent components; and [AC78], which studied decompositions with lossless joins and dependency preservation. This was extended independently to arbitrary decompositions over fd's by [BR80] and [MMSU80]. Lossless join was further investigated in [Var82b] (see Exercise 11.20).

The notion that not all integrity constraints specified in a schema should be considered for the design process was implicit in various works on semantic data modeling (e.g., [Che76, Lie80, Lie82]). It was stated explicitly in connection with relational schema design in [FMU82, Sci81]. An extensive application of this approach to develop an approach to schema design that incorporates both fd's and mvd's is [BK86].

A very different form of decomposition, called *horizontal decomposition*, is introduced in [DP84]. This involves splitting a relation into pieces, each of which satisfies a given set of fd's.

The universal relation assumption has a long history; the reader is directed to [AA93, MUV84, Ull89b] for a much more complete coverage of this topic than found in this chapter. The URA was implicit in much of the early work on normal forms and decompositions; this was articulated more formally in [FMU82, MUV84]. The weak URA was studied in connection with query processing in [Sag81, Sag83], and in connection with fd satisfaction in [Hon82]. Proposition 11.3.5(a) is due to [MUV84] and part (b) is due to [Hon82]; the extension to full dependencies is due to [GMV86]. Reference [Sci86] presents an interesting comparison of the relational model with inclusion dependencies to a variant of the universal relation model and shows an equivalence when certain natural restrictions are imposed.

A topic related to the URA is that of *universal relation interfaces* (URI); these attempt to present a user view of a relational database in the form of a universal relation. An excellent survey of research on this topic is found in [MRW86]; see also [AA93, Osb79, Ull89b].

Exercises

Exercise 11.1

(a) Extend the instance of Example 11.1.1 for **CINEMA-SEM** so that it has at least two objects in each class.

(b) Let **CINEMA-SEM**′ be the same as **CINEMA-SEM**, except that a complex value class *Movie_Actor* is used in **CINEMA-SEM** in place of the attributes *acted_in* and *has_actors*. How would the instance you constructed for part (a) be represented in **CINEMA-SEM**′?

Exercise 11.2

(a) Suppose that in **CINEMA-SEM** some theaters do not have phones. Describe how the simulation **CINEMA-REL** can be changed to reflect this (without using null values). What dependencies are satisfied?

(b) Do the same for the case where some persons may have more than one citizenship.

Exercise 11.3

(a) Describe a general algorithm for translating GSM schemas with keys into relational ones.

(b) Verify Theorem 11.1.3.

(c) Verify that the relational schema resulting from a GSM schema is in 4NF and has acyclic and key-based ind's.

♠ **Exercise 11.4** [MR88, MR92] Let $\mathbf{R}$ be a relational database schema, Σ a set of tagged fd's for $\mathbf{R}$, and Γ a set of ind's for $\mathbf{R}$. Assume that $(\mathbf{R}, \Sigma)$ is in BCNF and that Γ is acyclic and consists of key-based ind's (as will arise if $\mathbf{R}$ is the simulation of a GSM schema). Prove that Σ and Γ are independent. *Hint:* Show that if $\mathbf{I}$ is an instance of $\mathbf{R}$ satisfying Σ, then no fd can be applied during chasing of $\mathbf{I}$ by $(\Sigma \cup \Gamma)$. Now apply Theorem 9.4.5.

Exercise 11.5 [Fag79] Let (R, Σ) be a relation schema, and let Σ' be the set of key dependencies implied by Σ. Show that R is in 4NF iff each nontrivial mvd implied by Σ is implied by Σ'.

Exercise 11.6 [DF92] A key dependency $X \to U$ is *simple* if X is a singleton.

(a) Suppose that (R, Σ) is in BCNF, where Σ may involve both fd's and mvd's. Suppose further that (R, Σ) has at least one simple key. Prove that (R, Σ) is in 4NF.

(b) Suppose that (R, Σ) is in 3NF and that each key of Σ is simple. Prove that (R, Σ) is in BCNF.

A schema (R, Σ) is in *project-join normal form* (*PJ/NF*) if each JD σ implied by Σ is implied by the key dependencies implied by Σ.

(a) Show that if (R, Σ) is in 3NF and each key of Σ is simple, then (R, Σ) is in PJ/NF.

Exercise 11.7 Let (U, Σ) be a schema, where Σ contains possibly fd's, mvd's, and jd's. Show that (a) (U, Σ) is in BCNF implies (U, Σ) is in 3NF; (b) (U, Σ) is in 4NF implies (U, Σ) is in BCNF; (c) (U, Σ) is in PJ/NF implies (U, Σ) is in 4NF.

Exercise 11.8 [BR80, MMSU80] Prove Theorem 11.2.3.

Exercise 11.9 Recall the schema (*Movies*[*TDA*],$\{T \to D\}$). Consider the decomposition $\mathbf{R}_1 = \{(TD, \{T \to D\}), (DA, \emptyset)\}$.

(a) Show that this does not have the lossless join property.

★(b) Show that this decomposition is not one-to-one. That is, exhibit two distinct instances I, I' of (*Movies*, $\{T \to D\}$) such that $\pi_{\mathbf{R}_1}(I) = \pi_{\mathbf{R}_1}(I')$.

Exercise 11.10 Verify Theorem 11.2.8. *Hint:* To prove the lossless join property, use repeated applications of Proposition 8.2.2.

Exercise 11.11 [FJT83] For each $n \geq 0$, describe an fd schema (U, Σ) and $V \subseteq U$, such that Σ has $\leq 2n + 1$ dependencies but the smallest cover for $\pi_V(\Sigma)$ has at least 2^n elements.

Exercise 11.12

(a) Let $(U[Z], \Gamma)$ be an fd schema. Give a polynomial time algorithm for determining whether this relation schema is in BCNF. (In fact, there is a linear time algorithm.)

(b) [BB79] Show that the following problem is co-NP-complete. Given fd schema $(R[U], \Sigma)$ and $V \subseteq U$, determine whether $(V, \pi_V(\Sigma))$ is in BCNF. *Hint:* Reduce to the hitting set problem [GJ79].

★**Exercise 11.13** [TF82] Develop a polynomial time algorithm for finding BCNF decompositions. *Hint:* First show that each two-attribute fd schema is in BCNF. Then show that if $(S[V], \Omega)$ is not in BCNF, then there are $A, B \in V$ such that $(V - AB) \to A$.

Exercise 11.14 Recall the schema *Showings*[*Th*(*eater*), *Sc*(*reen*), *Ti*(*tle*), *Sn*(*ack*)] of Section 8.1, which satisfies the fd *Th*,*Sc* $\to$ *Ti* and the mvd *Th* $\twoheadrightarrow$ *Sc*,*Ti* | *Sn*. Consider the two decompositions

$$\mathbf{R}_1 = \{\{Th, Sc, Ti\}, \{Th, Sn\}\}$$
$$\mathbf{R}_2 = \{\{Th, Sc, Ti\}, \{Th, Sc, Sn\}\}.$$

Are they one-to-one? dependency preserving? Describe anomalies that can arise if either of these decompositions is used.

Exercise 11.15 [BB79] Verify that the schema of Example 11.2.10 has no BCNF decomposition that preserves dependencies.

Exercise 11.16 [Mai80] Develop a polynomial time algorithm that finds a minimal cover of a set of fd's.

Exercise 11.17 Prove Theorem 11.2.14.

Exercise 11.18 [Mai83] Show that a schema $(R[U], \Sigma)$ with $2n$ attributes and $2n$ fd's can have as many as 2^n keys.

Exercise 11.19 [LO78] Let $(S[V], \Omega)$ be an fd schema. Show that the following problem is NP-complete: Given $A \in V$, is there a nontrivial fd $Y \to A$ implied by Ω, where Y is not a superkey and A is not a key attribute?

★ **Exercise 11.20** [Var82b] For this exercise, you will exhibit an example of a schema (R, Σ), where Σ consists of dependencies expressed in first-order logic (which may not be embedded dependencies) and a decomposition $\mathbf{R}$ of R such that $\mathbf{R}$ is one-to-one but does not have the lossless join property.

Consider the schema $R[ABCD]$. Given $t \in I \in inst(R)$, $t[A]$ is a *key element* for AB in I if there is no $s \in I$ with $t[A] = s[A]$ and $t[B] \neq s[B]$. The notion of $t[C]$ being a *key element* for CD is defined analogously. Let Σ consist of the constraints

(i) $\exists t \in I$ such that both $t[A]$ and $t[C]$ are key elements.

(ii) If $t \in I$, then $t[A]$ is a key element or $t[C]$ is a key element.

(iii) If $s, t \in I$ and $s[A]$ or $t[C]$ is a key element, then the tuple u is in I, where $u[AB] = s[AB]$ and $u[CD] = t[CD]$.

Let $\mathbf{R} = \{R_1[AB], R_2[CD]\}$ be a decomposition of (R, Σ).

(a) Show that the decomposition $\mathbf{R}$ for (R, Σ) is one-to-one.

(b) Exhibit a reconstruction mapping for $\mathbf{R}$. (The natural join will not work.)

Exercise 11.21 This and the following exercise provide one kind of characterization of the relative information capacity of decompositions of relation schemas. Let U be a set of attributes, let $\alpha = \{X_1, \ldots, X_n\}$ be a nonempty family of subsets of U, and let $X = \cup_{i=1}^n X_i$. The *project-join* mapping determined by α, denoted PJ_α, is a mapping from instances over U to instances over $\cup_{i=1}^n X_i$ defined by $PJ_\alpha(I) = \bowtie_{i=1}^n (\pi_{X_i}(I))$. α is *full* if $\cup_{i=1}^n = U$, in which case PJ_α is a *full* project-join mapping.

Prove the following for instances I and J over U:

(a) $\pi_X(I) \subseteq PJ_\alpha(I)$

(b) $PJ_\alpha(PJ_\alpha(I)) = PJ_\alpha(I)$

(c) if $I \subseteq J$ then $PJ_\alpha(I) \subseteq PJ_\alpha(J)$.

★ **Exercise 11.22** [BMSU81] Let U be a set of attributes. If $\alpha = \{X_1, \ldots, X_n\}$ is a nonempty full family of subsets of U, then $Fixpt(\alpha)$ denotes $\{I$ over $U \mid PJ_\alpha(I) = I\}$ (see the preceding exercise). For α and β nonempty full families of subsets of U, β *covers* α, denoted $\alpha \preceq \beta$, if for each set $X \in \alpha$ there is a set $Y \in \beta$ such that $X \subseteq Y$. Prove for nonempty full families α, β of subsets of U that the following are equivalent:

(a) $\alpha \preceq \beta$

(b) $PJ_\alpha(I) \supseteq PJ_\beta(I)$ for each instance I over U

(c) $Fixpt(\alpha) \subseteq Fixpt(\beta)$.

Exercise 11.23 Given relational database schemas $\mathbf{S}$ and $\mathbf{S}'$, we say that $\mathbf{S}'$ *dominates* $\mathbf{S}$ using the calculus, denoted $\mathbf{S} \preceq_{\text{calc}} \mathbf{S}'$, if there are calculus queries $q : Inst(\mathbf{S}) \to Inst(\mathbf{S}')$ and $q' : Inst(\mathbf{S}') \to Inst(\mathbf{S})$ such that $q \circ q'$ is the identity on $Inst(\mathbf{S})$. Let schema $R = (ABC, \{A \to B\})$ and the decomposition $\mathbf{R} = \{(AB, \{A \to B\}), (AC, \emptyset)\}$. (a) Verify that $R \preceq_{\text{calc}} \mathbf{R}$. (b) Show that $\mathbf{R} \not\preceq_{calc} R$. *Hint:* For schemas $\mathbf{S}$ and $\mathbf{S}'$, $\mathbf{S}'$ *dominates* $\mathbf{S}$ *absolutely*, denoted $\mathbf{S} \preceq_{abs} \mathbf{S}'$, if there is some $n \geq 0$ such that for each finite subset $\mathbf{d} \subseteq \mathbf{dom}$ with $|\mathbf{d}| \geq n$, $|\{\mathbf{I} \in Inst(\mathbf{S}) \mid adom(\mathbf{I}) \subseteq \mathbf{d}\}| \leq |\{\mathbf{I} \in Inst(\mathbf{S}') \mid adom(\mathbf{I}) \subseteq \mathbf{d}\}|$. Show that $\mathbf{S} \preceq_{calc} \mathbf{S}'$ implies $\mathbf{S} \preceq_{abs} \mathbf{S}'$. Then show that $\mathbf{R} \not\preceq_{abs} R$.

★ **Exercise 11.24** [HY84] Let A and B be relational attributes. Consider the complex value type $T = \langle A, \{B\}\rangle$, where each instance of T is a finite set of pairs having the form $\langle a, \hat{b}\rangle$, where $a \in \mathbf{dom}$ and $\hat{b}$ is a finite subset of $\mathbf{dom}$. Show that for each relational schema $\mathbf{R}$, $\mathbf{R} \preceq_{abs} T$ and $T \not\preceq_{abs} \mathbf{R}$. (See Exercise 11.23 for the definition of $\preceq_{abs}$.)

♠ **Exercise 11.25** [BV84b, CP84]

(a) Let (U, Σ) be a (full dependencies) schema and $\mathbf{R}$ an acyclic decomposition of U (in the sense of acyclic joins). Then $\pi_{\mathbf{R}}$ is one-to-one iff $\mathbf{R}$ has the lossless join property. *Hint:* First prove the result for the case where the decomposition has two elements (i.e., it is based on an mvd). Then generalize to acyclic decompositions, using an induction based on the GYO algorithm.

(b) [CKV90] Show that (a) can be generalized to include unary ind's in Σ.

Exercise 11.26 [Hon82] Let (U, Σ) be an fd schema and $\mathbf{R}$= $\{R_1, \ldots, R_n\}$ a decomposition of U. Consider the following notions of "satisfaction" by $\mathbf{I}$ over $\mathbf{R}$ of Σ:

$\mathbf{I} \models_1 \Sigma$**:** if $I_j \models \pi_{R_j}(\Sigma)$ for each $j \in [1, n]$.
$\mathbf{I} \models_2 \Sigma$**:** if $\bowtie \mathbf{I} \models \Sigma$.
$\mathbf{I} \models_3 \Sigma$**:** if $\mathbf{I} = \pi_{\mathbf{R}}(I)$ for some I over U such that $I \models \Sigma$.

(a) Show that $\models_1$ and $\models_2$ are incomparable.

(b) Show that if $\mathbf{R}$ preserves dependencies, then $\models_1$ implies $\models_2$.

(c) What is the relationship of $\models_1$ and $\models_2$ to $\models_3$?

(d) What is the relationship of all of these to the notion of satisfaction based on the weak URA?

♠ **Exercise 11.27** [Hon82] Prove Theorem 11.3.4.

Exercise 11.28 [MUV84, Hon82] Prove Proposition 11.3.5.