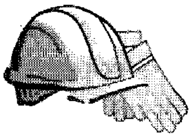

19

SCHEMA REFINEMENT AND NORMAL FORMS

- What problems are caused by redundantly storing information?
- What are functional dependencies?
- What are nornlal forms and what is their purpose?
- What are the benefits of BCNF and 3NF?
- What are the considerations in decolllposing relations into appropriate normal forms?
- Where does normalization fit in the process of database design?
- Are luore general dependencies useful in database design?
- Key concepts: redundancy, insert, delete, and update anomalies; functional dependency, Armstrong's Axioms; dependency closure, attribute closure; normal fonns, BCNF, 3NF; decOlnpositions, lossless-join, dependency-preservation; multivalued dependencies, join dependencies, inclusion dependencies, 4NF, 5NF

> It is a nlelancholy truth that even great InelI have their poor relations.
>
> Charles Dickens

Conceptual database design gives us a set of relation schemas and integrity constraints (ICs) that can be regarded as a good starting point for the final database design. T'his initial design IHust be refined by taking the lCg into account rnore fully than is possiblc\vith just the ER rnodel constructs aIId also by considering perforrnance criteria and typical workloads. In this chapter, we c!iscllss how lCs can be used to refine the conceptual schema produced by translating an ER 1Hodel design into a collection of relations. \Vorkload and performance considerations are discussed in Chapter 20.

We concentrate on an irnportant class of constraints called *functional dependencies*. Other kinds of les, for exarnple, *multivalued dependencies* and *join dependencies*, also provide useful inforrnation. They can sOIuetilnes reveal redundancies that cannot be detected using functional dependencies alone. We discuss these other constraints briefly.

This chapter is organized as follows. Section 19.1 is an overview of the schenla refineInent approach discussed in this chapter. We introduce functional dependencies in Section 19.2. In Section 19.3, we show how to reason with functional dependency information to infer additional dependencies from a given set of dependencies. We introduce norlnal forIns for relations in Section 19.4; the normal form satisfied by a relation is a measure of the redundancy in the relation. A relation with redundancy can be refined by *decomposing it,* or replacing it with smaller relations that contain the saIne information but without redundancy. We discuss deco1npositions and desirable properties of decompositions in Section 19.5, and we show how relations can be decomposed into smaller relations in desirable normal forms in Section 19.6.

In Section 19.7, we present several examples that illustrate how relational schemas obtained by translating an ER model design can nonetheless suffer froln redundancy, and we discuss how to refine such schemas to eliminate the problems. In Section 19.8, we describe other kinds of dependencies for database design. We conclude with a discussion of nornlalization for our case study, the Internet shop, in Section 19.9.

19.1 INTRODUCTION TO SCHEMA REFINEMENT

We now present an overview of the probleIns that schenla refinement is intended to address and a refinernent approach based on decolnpositions. Iledundant storage of inforrnation is the root cause of these problerns. Although decoInposition can elirninate redundancy, it can lead to problclns of its own and should be used with caution.

19.1.1 Problems Caused by Redundancy

Storing the same inforrnation redundantly, that is, in l110re than one place \vithin a database, can lead to several problcll1S:

- Redundant Storage: SOU1C iuforInation is stored repeatedly.

- Update Anomalies: If one copy of sueh repeated data is updated, an inconsistency is created unless all copies are sirnilarly updated.
- Insertion Anomalies: It may not be possible to store certain inforlnation unless sorne other, unrelated, inforIIlatioIl is stored as well.
- Deletion Anomalies: It rnay not be possible to delete certain inforrnation without losing SOHle other, unrelated, infofrnation as well.

Consider a relation obtained by translating a variant of the IIourly_Emps entity set frorn Chapter 2:

Hourly_Emps(<u>*ssn*</u>, *name*, *lot*, *rating*, *hourly_wages*, *hours_worked*)

In this chapter, we ornit attribute type inforrnation for brevity, since our focus is on the grouping of attributes into relations. We often abbreviate an attribute narne to a single letter and refer to a relation schema by a string of letters, one per attribute. For exarllple, we refer to the Hourly_Ernps scherna as *SNLRWH* (*W* denotes the *hourly_wages* attribute).

1'he key for Hourly_Emps is *ssn*. In addition, suppose that the *hourly_wages* attribute is deterrnined by the *rating* attribute. That is, for a given *rating* value, there is only one perrllissible *hourly_wages* value. This IC is an exanlple of a *functional dependency*. It leads to possible redundancy in the relation Hourly_Ernps, as illustrated in Figure 19.1.

	narne	*lot*	*rating*	*hourly_wages*	*hours_worked*
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Sruiley	22	8	10	30
131-24-3650	Srllethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Figure 19.1 An Instance of the Hourly_Emps Relation

If the same value appears in the *rating* colurnn of two tuples, the IC tells us that the sarne value HUlst appear in the *hourly_wages* colurnn as well. This redundancy has the sarne negative consequences as before:

- *Redundant Storage:* rrhe rating value 8 corresponds to the hourly wage 10, and this association is repeated three tirnes.
- *[Tpdate Anomalies:* The *hourly_wages* in the first tuple could be updated without rnaking a sirnilar change in the second tuple.

- *Insertion Anomalies:* We cannot insert a tuple for an crnployee unless \ve know the hourly wage for the ernployee's rating value.
- *Delet'ion Anomalies:* If we delete all tuples with a given rating value (e.g., we delete the tuples for Snlcthurst and Guldu) we lose the association between that *rating* value and its *hourly_wage* value.

Ideally, we want schemas that do not pennit redundancy, but at the very least we want to be able to identify schernas that do allow redundancy. Even if we choose to accept a scherna with sorne of these drawbacks, perhaps owing to perforlnance considerations, we want to rnake an infonned decision.

Null Values

It is worth considering whether the use of *null* values can address some of these problems. As we will see in the context of our exarnple, they cannot provide a complete solution, but they can provide sorne help. In this chapter, we do not discuss the use of *null* values beyond this one exarnple.

Consider the example Hourly_Elnps relation. Clearly, *null* values cannot help eliminate redundant storage or update anomalies. It appears that they can address insertion and deletion anomalies. For instance, to deal with the insertion anolnaly exarnple, we can insert an elTIplayee tuple with *null* values in the hourly wage field. However, *null* values cannot address all insertion anornalies. For exarnple, we cannot record the hourly wage for a rating unless there is an ernployee with that rating, because we cannot store a null value in the *ssn* field, which is a prirnary key field. Sinlilarly, to deal with the deletion anomaly exarnple, we rnight consider storing a tuple with *null* values in all fields except *rating* and *hourly_wages* if the last tuple with a given *rating* would otherwise be deleted. However, this solution does not work because it requires the *8871,* value to be *null,* and prirnary key fields cannot be *null.* Thus, *null* values do not provide a general solution to the problerns of reclundancy, even though they can help in sorne cases.

19.1.2 Decompositions

Intuitively, redundancy arises when a relational schcrna forces an association between attributes that is not natural. Functional dependencies (and, for that matter, other Ies) can 'be used to identify such situations and suggest re£1nernents to the schema. The essential idea is that rnany problerns arising fr0111 redundancy can be addressed by replacing a relation 'with a collection of 'smaller' relations.

A decomposition of a relation schema R consists of replacing the relation schema by two (or more) relation schemas that each contain a subset of the attributes of R and together include all attributes in R. Intuitively, we want to store the information in any given instance of R by storing projections of the instance. This section examines the use of decompositions through several examples.

We can decompose Hourly_Emps into two relations:

Hourly_Emps2(*ssn*, *name*, *lot*, *rating*, *hours_worked*)
Wages(*rating*, *hourly_wages*)

The instances of these relations corresponding to the instance of Hourly_Emps relation in Figure 19.1 is shown in Figure 19.2.

ssn	*name*	*lot*	*rating*	*hours_worked*
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

rating	*hourly_wages*
8	10
5	7

Figure 19.2 Instances of Hourly_Emps2 and Wages

Note that we can easily record the hourly wage for any rating simply by adding a tuple to Wages, even if no employee with that rating appears in the current instance of Hourly_Emps. Changing the wage associated with a rating involves updating a single Wages tuple. This is more efficient than updating several tuples (as in the original design), and it eliminates the potential for inconsistency.

19.1.3 Problems Related to Decomposition

Unless we are careful, decomposing a relation schema can create more problems than it solves. Two important questions must be asked repeatedly:

1. Do we need to decompose a relation?

2. What problerns (if any) does a given deeornposition cause?

To help with the first question, several *normal forms* have been proposed for relations. If a relation scherna is ill one of these nOfrual 1'orrns, we know that certain kinds of problerlls cannot arise. Considering the norrnal forrn of a given relation scherna can help us to decide \vhether or not to decornpose it further. If we decide that a relation scherna 111USt be decornposed further, we rnust choose a particular dec()lnposition (I.e., a particular collection of slnaller relations to replace the given relation).

With respect to the second question, two properties of decornpositions are of particular interest. The *lossless-join* property enables us to recover any instance of the decornposed relation froln corresponding instances of the s111aller relations. The *dependency-preservation* property enables us to enforce any constraint on the original relation by sinlply enforcing SaIne contraints on each of the srnaller relations. That is, we need not perform joins of the slllaller relations to check whether a constraint on the original relation is violated.

From a performance standpoint, queries over the original relation may require us to join the decomposed relations. If such queries are common, the perfor-rnance penalty of decomposing the relation may not be acceptable. In this case, we may choose to live with some of the problems of redundancy and not decompose the relation. It is important to be aware of the potential problerns caused by such residual redundancy in the design and to take steps to avoid thern (e.g., by adding SaIne checks to application code). In sonle situations, decomposition could actually *improve* performance. This happens, for exam-ple, if lnost queries and updates exanline only one of the decornposed relations, which is srnaller than the original relation. We do not discuss the irnpact of decompositions on query perforInance in this chapter; this issue is covered in Section 20.8.

()ur goal in this chapter is to explain S0111e powerful concepts and design guide-lines based on the theory of functional dependencies. A good database designer should have a firm grasp of nor1nal fonns and \vhat problerns they (do or do not) alleviate, the technique of decornposition, and potential problerns with decornpositions. For example, a designer often asks questions such as these: Is a relation in a given nonnal forIn? Is a decornposition clependency-preserving? Our objective is to explain when to raise these questions and the significance of the answers.

19.2 FUNCTIONAL DEPENDENCIES

A functional dependency (FD) is a kind of Ie that generalizes the concept of a *key*. Let R be a relation scherna and let X and Y be nonernpty sets of attributes in R. We say that an instance r of R satisfies the FDX $\rightarrow Y$ 1 if the following holds for every pair of tuples tl and $t2$ in r:

If $t1.X$ = $t2..X$, then $tl.}T$ = $t2.Y$.

We use the notation tl.X to refer to the projection of tuple t1 onto the attributes in X, in a natural extension of our TRC notation (see Chapter 4) $t.a$ for referring to attribute a of tuple t. An FD $X \rightarrow$ Yessentially says that if two tuples agree on the values in attributes X, they 111Ust also agree on the values in attributes Y.

Figure 19.3 illustrates the rneaning of the FD $AB \rightarrow$ C by showing an instance that satisfies this dependency. The first two tuples show that an FD is not the same as a key constraint: Although the FD is not violated, AB is clearly not a key for the relation. The third and fourth tuples illustrate that if two tuples differ in either the A field or the B field, they can differ in the C field without violating the FD. On the other hand, if we add a tuple (aI, bl, *c2, dl)* to the instance shown in this figure, the resulting instance would violate the FD; to see this violation, compare the first tuple in the figure with the new tuple.

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	dl
a2	bl	c3	ell

Figure 19.3 An Instance that Satisfies $AB \rightarrow C$

Recall that a *legal* instance of a relation nUlst satisfy all specified les, including all specified FDs. As noted in Section 3.2, Ies rIlust be identified and specified based on the sernantics of the real-world enterprise being n1odeled. By looking at an instance of a relation, we rnight be able to tell that a certain FD does *not* hold. I-Iowever; we can never deduce that an FD *docs* hold by looking at one or 1I10re instances of the relation, beca,use an FD, like other les, is a staternent about *all* possible legal instances of the relation.

1$X \rightarrow Y$ is read as X *functionally determines* Y, or simply as X *determines* Y.

.A prirnary key constraint is a special case of an FD. The attributes in the key play the role of X, and the set of all attributes in the relation plays the role of Y. Note, however, that the definition of an FD does not require that the set X be 111iniInal; the additionalrninimality condition Illust be Inet for X to be a key. If $X \rightarrow Y$ holds, \vhere Y is the set of all attributes, and there is SCHne (strictly contajned) subset V of X such that $V \rightarrow Y$ holds, then X is a *superkey.*

In the rest of this chapter, ·we see several exarIlples of FDs that are not key constraints.

19.3 REASONING ABOUT FDS

Given a set of FDs over a relation scheula .R, typically several additional FDs hold over R whenever all of the given FDs hold. As an exalnple, consider:

> Workers(*ssn*, *naTne*, *lot*, *did*, *since*)

We know that *ssn* → *did* holds, since *ssn* is the key, and FD *did* → *lot* is given to hold. Therefore, in any legal instance of Workers, if two tuples have the same *ssn* value, they Blust have the sarne *did* value (frolH the first FD), and because they have the sarrle *did* value, they must also have the saIne *lot* value (1'1'0111 the second FD). Therefore, the FD *ssn* → *lot* also holds on Workers.

We say that an FD f is implied by a given set F of FDs if f holds on every relation instance that satisfies all dependencies in F; that is, f holds whenever all FDs in F hold. Note that it is not sufficient for f to hold on SaIne instance that satisfies all dependencies in F; rather, f rnust hold on *every* instance that satisfies all dependencies in F.

19.3.1 Closure of a Set of FDs

The set of all .FDs irnplied by a given set F of FDs is called the closllre of **F**, denoted as F^+. An irnportant question is how we can infer, or cornpute, the closure of a given set F of FDs. The answer is sirnple and elegant. The following three rules, called Armstrong's Axioms, can be applied repeatedly to infer all FI)s irnplied by a set F of FDs. We use X, Y, and Z to denote *sets* of attributes over a relation scherna R:

- Reflexivity: If $X \supseteq Y$, then $X \rightarrow Y$.
- Augn1.entation: If)($\rightarrow Y$, then $XZ \rightarrow YZ$ for any Z.
- Transitivity: If)($\rightarrow Y$ and Y Z, then $X \rightarrow Z$.

Theorem 1 *Armstrong's Axioms are* **sound**, *in that they generate only FDs in* F^+ *when applied to a set* F *of FDs. They are also* **complete**, *in that repeated a]Jplicat'ion afthese rules will generate all FDs in the closure* .Fl+.

The soundness of Arrnstrong's Axiorns is straightfor\vard to prove. Cornpleteness is harder to show; see Exercise 19.17.

It is convenient to use SOlne additional rules while reasoning about *P+:*

- Union: If $X \rightarrow Y$and $X \rightarrow Z$, then $X \rightarrow YZ$.
- Decomposition: If $X \rightarrow YZ$, then $X \rightarrow y'$ and $X \rightarrow Z$.

These additional rules are not essential; their soundness can be proved using Armstrong's AxiollIS.

To illustrate the use of these inference rules for FDs, consider a relation schelua ABC with FDs $A \rightarrow B$ and $B \rightarrow C$. In a trivial FD, the right side contains only attributes that also appear on the left side; such dependencies always hold due to reflexivity. Using reflexivity, we can generate all trivial dependencies, which are of the form:

$$X \rightarrow Y, \text{ where } Y \subseteq X,\ X \subseteq ABC, \text{ and } Y \subseteq ABC.$$

FrOHl transitivity we get $A \rightarrow C$. Fronl auglnentation we get the nontrivial dependencies:

$$AC \rightarrow BC,\ AB \rightarrow AC,\ AB \rightarrow C13.$$

As another exalnple, we use a rnore elaborate version of Contracts:

Contracts(*contractid*, *supplierid*, *projectid*, *deptid*, *partid*, *qty*, *val'ue)*

\Ve denote the schenla for Contracts as *CSJDPQV.* The rneaning of a tuple is that the contract with *contractid* C is an agreement that supplier *S* (*supplierid*) 'will supply *Q* iterns of part? *(par-tid)* to project J (*projectid*) associated with departrnent *D* (*deptid*); the value *V* of this contract is equal to *value.*

The following res are known to hold:

1. The contract id C is a key: $C \rightarrow CSJDP(JV$.
2. A project purchases a given part using a single contract: $.II) \rightarrow C$.

3. A departInent purchases at most one part froul a supplier: $8D \rightarrow P$.

Several additional FDs hold in the closure of the set of given FDs:

From $.IP \rightarrow C$, $C \rightarrow CSJDPQV$, and transitivity, we infer $.IP \rightarrow CSJDPQV$.

FraIn $8D \rightarrow P$ and augnlentation, we infer $SDJ \rightarrow JP$.

FraIn $8DJ \rightarrow .IP$, $JP \rightarrow CSJDPQV$, and transitivity, we infer $SDJ \rightarrow CSJDPQV$. (Incidentally, while it Illay appear tenlpting to do so, we *cannot* conclude $SD \rightarrow CSDPQV$, canceling $.I$ on both sides. FD inference is not like aritlunetic Illultiplication!)

We can infer several additionalFDs that are in the closure by using augruentation or decomposition. For exarnple, from $C \rightarrow CSJDPQV$, using decomposition, we can infer:

$$C \rightarrow C,\ C \rightarrow 5,\ C \rightarrow J,\ C \rightarrow D, \text{ and so forth}$$

Finally, we have a number of trivial FDs from the reflexivity rule.

19.3.2 Attribute Closure

If we just want to check whether a given dependency, say, $X \rightarrow Y$, is in the closure of a set F of FDs, we can do so efficiently without cornputing Fl+. We first cornpute the **attribute closure** X+with respect to F, \which is the set of attributes A such that $X \rightarrow A$ can be inferred using the Arrnstrong Axioms. The algorithrn for computing the attribute closure of a set X of attributes is shown in Figure 19.4.

```
closure = X;
repeat until there is no change: {
        if there is an FD U → V in F such that U ⊆ closure,
            then set closure == closure U V·
        }
```

Figure 19.4 Computing the Attribute Closure of Attribute Sct X

Theorem 2 *The algorithln shown in Figure 1.9.4 cornputes the attribute closure X-+- of the attribute set X 'IDith respect to the sct of FDs Fl.*

The proof of this theorern is considered in Exercise 19.15. This algoriUuIl can be rl10dified to find keys by starting with set X containing a, single attribute and stopping as soon as *closure* contains all attributes in the relation scherna. By varying the starting attribute and the order in which the algorithrIl considers FDs, we can obtain all candidate keys.

19.4 NORMAL **FORMS**

Given a relation sche111a, we need to decide whether it is a good design or we need to decornpose it into srnaller relations. Such a decision llUlst be guided by an understanding of what problenls, if any, arise froln the current schelna. To provide such guidance, several normal forms have been proposed. If a relation schelna is in one of these norrnal forIns, we know that certain kinds of problerlls cannot arise.

The nonnal forrns based on FDs are *first nor-rnal forrn* (1*NF)*, *second normal forrn (2NF)*, *third normal form (3NF)*, and *Boyce-Codd normal form (BCNF)*. These fonns have increasingly restrictive requirernents: Every relation in BCNF is also in 3NF, every relation in 3NF is also in 2NF, and every relation in 2NF is in INF. A relation is in first normal fortH if every field contains only atornic values, that is, no lists or sets. This requirerllent is iInplicit in our definition of the relational rnode!. Although SOHle of the newer database systerlls are relaxing this requirernent, in this chapter we aSSUlne that it always holds. 2NF is Inainly of historical interest. 3NF and BCNF are irnportant frolH a database design standpoint.

While studying norrnal fonns, it is irnportant to appreciate the role played by FDs. Consider a relation scherna R with attributes ABC. In the absence of any ICs, any set of ternary tuples is a legal instance and there is no potential for redundancy. ()n the other hand, suppose that we have the FI) $A \rightarrow$ 13. Now if several tuples have the sarne A value, they rnust also have tllC sarneB value. This potential redundancy can be predicted using the FD il1fonnation. If 11101'8 detailed 1Cs are specified, we rnay be able to detect rnore subtle redundancies as well.

We primarily discuss redundancy revealed l)y PI) inforrnation. In Section 19.8, we discuss 11lore sophisticated 1Cs called *multivalued dependencies* and *join dependencies* and norrnal forrns based on theIn.

19.4.1 Boyce...Codd Normal Form

Let R be a relation scherna, F be the set ofF'I)s given to hold over R, X be a subset of the attributes ofR, and A be an attribute of R. R is in Boyce-Codd normal form if, for everyFl)X $\rightarrow$ A in F, one of the follo\ving statements is true:

- $A \in X$; that is, it is a trivial FD, or
- X is a superkey.

Intuitively, in a BCNF relation, the only nontrivial dependencies are those in 'which a key detennines SaIne attribute(s). Therefore, each tuple can be thought of as an entity or relationship, identified by a key and described by the reluaining attributes. !(ent (in [425]) puts this colorfully, if a little loosely: “Each attribute nlust describe [an entity or relationship identified by] the key, the \vhole key, and nothing but the key." If we use ovals to denote attributes or sets of attributes and draw arcs to indicate FDs, a relation in BCNF has the structure illustrated in Figure 19.5, considering just one key for simplicity. (If there are several candidate keys, each candidate key can play the role of KEY in the figure, with the other attributes being the ones not in the chosen candidate key.)

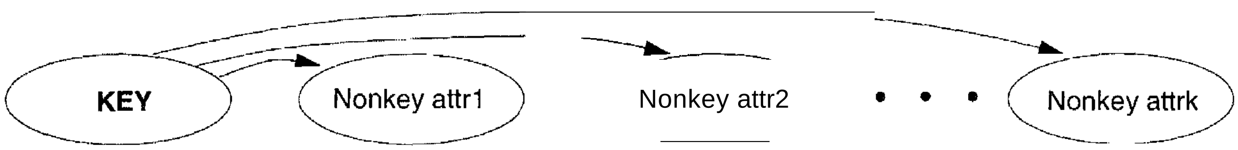

Figure 19.5 FDs in a BCNF Relation

BCNF ensures that no redundancy can be detected using FD infonnation alone. It is thus the Inost desirable norrnal form (fronl the point of view of redundancy) if we take into account only FD information. This point is illustrated in Figure 19.6.

X	Y	A
x	y_1	a
x	y_2	?

Figure 19.6 Instance Illustrating BCNF

This figure shows (t\VO tuples in) an instance of a relation with three attributes X, Y, and A. There a.re two tuples with the saIne value in the X colurnn. Now suppose that we kno\v that this instance satisfies an FD $X \rightarrow A$. We can see that one of the tuples has the value a in the A colurnn. What can we infer al)out the value in the A collllrnn in the second tuple? 'Using the FI), \ve can conclude that the second tuple also has the value a in this colurnn. (Note that this is really the only kind of inference we can make about values in the fields of tuples by usingFDs.)

But is this situation not an exaInple of redundancy? \Ve appear to have stored the value *a* twice. Can such a situation arise in a BCNF relation? The ans\ver is No! If this relation is in BCNF, because *A* is distinct fronl *X*, it follows that *X* IllU8t be a key. (Otherwise, the FD $X \rightarrow A$ \vould violate BCNF.) If *X* is a key, then Yl = *Y2*, which Ineans that the two tuples are identical Since a relation is defined to be a *set* of tuples, we cannot have two copies of the saIne tuple and the situation shc)\vn in Figure 19.6 cannot arise.

rrherefore, if a relation is in BCNF, every field of every tuple records a piece of inforlnation that cannot be inferred (using only FDs) frorn the values in all other fields in (all tuples of) the relation instance.

19.4.2 **Third Normal Form**

Let *R* be a relation scherna, *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 forIn if, for every FD $X \rightarrow A$ in *F*, one of the following statenlents is true:

- $A \in X$; that is, it is a trivial FD, or
- *X* is a superkey, or
- *A* is part of sorne key for *R*.

rrhe definition of 3NF is sinlilar to that of BCNF, with the only difference being the third condition. Every BCNF relation is also in 3NF. To understand the third condition, recall that a key for a rela,tion is a *minimal* set of attributes that uniquely deterrnines all other attributes. *A* rrlllst be part of a key (any key, if there are several). It is not enough for *A* to be part of a superkey, because the latter condition is satisfied by every attribute! Finding all keys of a relation scherna is known to be an NP-cornplete problern, and so is the prob1ern of detennining whether a relation seherna is in 3NF.

Suppose that a dependency $X \rightarrow A$ causes a violation of 3NF. There are two cases:

- X is *a proper subset of some key* K. Such a dependency is 801netirnes called a partial dependency. In this case, we store *(X, ./1)* pairs redundantl:y. As an example, consider the Reserves relation with attributes *SBDC* frorn Section 19.7.4. The only key is *8El)*, and we have the FD $S \rightarrow C$. We store the credit ca,rd nurnber for a sailor as lnany tirnes as there are reservations for that sailor.
- X is *not a proper subset of any key.* Such a dependerlcy is sornetirnes called a transitive **dependency**, because it rneans we have a chain of

dependencies $K \rightarrow X \rightarrow A$. The problem is that we cannot associate an X value with a K value unless we also associate an A value with an X value. As an exanlple, consider the Hourly_Emps relation with attributes *SNLRWH* froIn Section 19.7.1. The only key is S, but there is an FD $R \rightarrow W$, which gives rise to the chain $S \rightarrow R \rightarrow W$. The consequence is that we cannot record the fact that elnployee S has rating R without knowing the hourly \vage for that rating. This condition leads to insertion, deletion, and update anoIllalies.

Partial dependencies are illustrated in Figure 19.7, and transitive dependencies are illustrated in Figure 19.8. Note that in Figure 19.8, the set X of attributes 11lay or Illay not have some attributes in conunon with KE-Y; the diagranl should be interpreted as indicating only that X is not a subset of KEY.

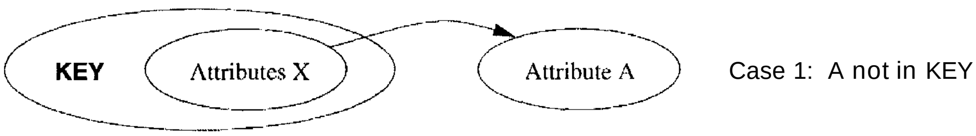

Figure 19.7 Partial Dependencies

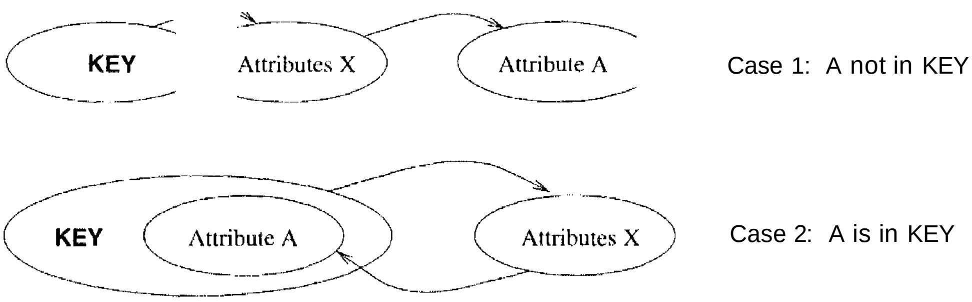

Figure 19.8 Transitive Dependencies

The Inotivation for 3NF is rather technical. By Inaking an exception for certain dependencies involving key attributes, we can ensure that every relation schclna can be decornposed into a collection of 3NF relations using only dec(nnpositions that have certain desirable properties (Section 19.5). Such a guarantee does not exist for BCNF relations; the 3NF definition weakens the BCNF requirernents just enough to Inake this guarantee possible. We Inay therefore cOlnprornise by settling for a 3NF design. As we see in Chapter 20, we 11lay sometimes accept this compromise (or even settle for a non-3NF scheIna) for other reasons as well.

lJnlike BCNF, however, BOlne redundancy is possible with 3NF. The problerns associated \vith partial and transitive dependencies persist if there is a nontrivial dependency $X \rightarrow A$ and X is not a superkey, even if the relation is in 3NF l)ccause A is pa,rt of a key. To understand this point, let us revisit the R,eserves relation with attributes *SEDe* and the FD $S \rightarrow C$, which states that a sailor uses a unique credit card to pay for reservations. S is not a key, and C is not part of a key. (In fact, the only key is *SED.)* Hence, this relation is not in 3NF; *(S, CJ* pairs are stored redundantly. IIowever, if we also know that credit cards uniquely identify the owner, we have the FD C $\rightarrow$ 5, which rneans that *CBD* is also a key for Reserves. Therefore, the dependency $S \rightarrow C$ does not violate 3NF, and R,eserves is in 3NF. Nonetheless, in all tuples containing the saIne 5 value, the saIne (8, *CJ* pair is redundantly recorded.

For cOlllpleteness, we reluark that the definition of second norrnal form is essentially that partial dependencies are not allowed. Thus, if a relation is in 3NF (which precludes both partial and transitive dependencies), it is also in 2NF.

19.5 PROPERTIES OF DECOMPOSITIONS

DecoIllposition is a tool that allows us to eliminate redundancy. As noted in Section 19.1.3, however, it is important to check that a decoInposition does not introduce new problellls. In particular, we should check whether a decomposition allows us to recover the original relation, and whether it allows us to check integrity constraints efficiently. We discuss these properties next.

19.5.1 Lossless-Join Decomposition

Let R be a relation schelna and let F be H, set of FDs over R. A decolnposition of R into two schernas with attribute sets X andY is said to be a lossless-join decomposition with respect to F if, for every instance r of R that satisfies the dependencies in F, $\pi_X(r) \bowtie \pi_Y(r) = T$. In other words, we can recover the original relation 1'rom the deconlposed relations.

This definition can easily be extended to cover a decornposition of R into more than two relations. It is easy to see that $T \subseteq \pi_X(r) \bowtie \pi_Y(r)$ ahvays holds. III general, though, the other direction does not hold. If we take projections of a relation and recornbine theln using natural join, we typically obta.in sOlne tuples that 'were not in the original relation. This situation is illustrated in Figure 19.9.

By replacing the instance r shown in Figure 19.9 with the instances $\pi_{SP}(r)$ and $\pi_{PI)}(r)$, we lose sorne information. In particular, suppose that the tuples in r denote relationships. We can no longer tell that the relationships $(81, p_1, d_3)$ and (s_3, p_1, d_1) do not hold. rrhe decoluposition of schema *SPD* into *S.P* and *PI)* is therefore loss,Y if the instance r shown in the figure is legal, that is, if this

S	P	D
s1	pl.	ell
s2	p2	d2
s3	pl.	d3

Instance r

S	P
s1	pI
s2	p2
s3	pI

$\pi_{SP}(r)$

P	D
p1	d1
p2	d2
~~p1~~	~~d3~~

$\pi_{PD}(r)$

S	P	D
81	pI	ell
82	p2	d2
83	pI	d3
sl	pI	d3
s3	pI	ell

$\pi_{SP}(r) \bowtie \pi_{PD}(r)$

Figure 19.9 Instances Illustrating Lossy Decompositions

instance could arise in the enterprise being rIlodeled. (Observe the siInilarities between this example and the Contracts relationship set in Section 2.5.3.)

All decompositions used to eli'minate redundancy **must** *be lossless.* The following sirnple test is very useful:

Theorern 3 *Let* R *be a relation and* F *be a set of FDs that hold over* R. *The decomposition of* R *into relations with attribute sets* R_1 *and* R_2 *is l08sless if and only if* $p+$ *contains either the FD* R_1 n $R2 \rightarrow R_1$ *or the FD* R_1 n $R_2 \rightarrow R2$.

In other words, the attributes cornrIlon to Rl and $R2$ HUlst contain a key for either RIOI' R_2.[2] If a relation is decornposed into 1110re than two relations, an efficient (time polynomial in the size of the dependency set) algoritllln is available to test whether or not the dec(nnposition is lossless, but we will not discuss it.

Consider the lIourly_Ernps relation again. It has attributes *SNLRWII*, and the FI) $R \rightarrow W$ causes a violation of 3NF. We dealt with this violation by decorIlposing the relation into *SNLRII* and *RW*. Since R is cornrnon to both decornposed relations and $R \rightarrow W$ holds, this decornposition is lossless-join.

This exarnple illustrates a general observation that follows froIH Theorerll 3:

> If an Ff) $X \rightarrow Y$ holds over a relation R and X n Y is ernpty, the decornposition of R into $R - Y$ and XY is lossless.

X appears in both $R - Y$ (since $X \cap Y$ is ernpty) and XY, and it is a key for XY.

[2]See Exercise 19.19 for a proof of Theorern 3. Exercise 19.11 illustrates that the 'only if' claim depends on the assumption that only functional dependencies can be specified as integrity constraints.

Another hnportant observation, which we state without proof, has to do with repeated decolnpositiollS. Suppose that a relation R is decornposed into Rl and $R2$ through a IOBsless-join decolupositiol1, and that $R1$ is decolnposed *intoRl.1* and $R12$ through another lossless-join decolnposition. Then, the decolnposition of R into $R11$, $R.12$, and $R2$ is lossless-join; by joining $R11$ and $R12$, \ve can recover $R.1$, and by then joining Rl and $R2$, we can recover R.

19.5.2 Dependency-Preserving Decomposition

Consider the Contracts relation with attributes *C8JDPCJVfronl* Section 19.3.l. The given FDs are $C \rightarrow C8JDPQV$, $JP \rightarrow C$, and $SD \rightarrow P$. Because SD is not a key the dependency $SD \rightarrow P$ causes a violation of BCNF.

We can decolnpose Contracts into two relations with schelnas $CSJDQV$ and SDP to address this violation; the decolnposition is lossless-join. There is one subtle problelll, however. We can enforce the integrity constraint $JP \rightarrow C$ easily when a tuple is inserted into Contracts by ensuring that no existing tuple has the same JP values (as the inserted tuple) but different C values. Once we decompose Contracts into $CSJDQV$ and SDP, enforcing this constraint requires an expensive join of the two relations whenever a tuple is inserted into $CSJDQV$. We say that this decornposition is not dependency-preserving.

Intuitively, a *dependency-preserving decornposition* allows us to enforce all FDs by exarnining a single relation instance on each insertion or rnodification of a tuple. (Note that deletions cannot cause violation of FDs.) To define dependency-preserving decornpositions precisely, we have to introduce the concept of a projection of FDs.

Let R be a relation schenla that is decolnposed into two schemas with attribute sets X' and Y, and let F be a set of FDs over R. The **projection of F on X** is the set of FDs in the closure $l^{i'}+$ (not just $.F$!) that involve only attributes in X. We denote the projection of F on attributes X as F_X. Note that a dependency $U \rightarrow V$ in $F+$ is in F_X only if *all* the attributes in $[/$ and V are in X.

The decornposition of relation scherna R with FI)s F into schcrnas with attribute sets X and Y is dependency-preserving if $(F_X \cup F_Y)^+ = F^+$. That is, if we take the dependencies in F_X and F_Y and cornpute the closure of their un.ion, we get back all dependencies in the closure of F. rrherefore, we need to enforce onl,y the dependencies in F_X and F_Y; allFDs in F^+ are then sure to be satisfied. To enforce F_X,\vs need to exarnine only relation)((on in.serts to that relation). To enforce F_Y, we need to exarnine only relation Y.

To appreciate the need to consider the closure F^+ while COIUpllting the projection of F, suppose that a relation R with attributes ABC is decomposed into relations\vith attributes AB and Be:. The set F of FDs overR includes $A \rightarrow B$, $B \rightarrow C$, and $C \rightarrow A$. Of these, $A \rightarrow B$ is in F_{AB} and $B \rightarrow C$ is in F_{BC}. But is this decoIIlposition dependency-preserving? What about $C \rightarrow A$? This dependency is not irnplied by the dependencies listed (thus far) for F_{AB} and F_{BC}.

The closure of F contains all dependencies in F plus $A \rightarrow C$, $B \rightarrow A$, and $C \rightarrow B$. Consequently, F_{AB} also contains $B \rightarrow A$, and F_{BC} contains $C \rightarrow B$. Therefore, FAB U F_{BC} contains $A \rightarrow B$, $B \rightarrow C$, $B \rightarrow A$, and $C \rightarrow B$. The closure of the dependencies in F_{AB} and F_{BC} now includes $C \rightarrow A$ (which follows frorn $C \rightarrow B$, $B \rightarrow A$, and transitivity). l'hus, the deccHnposition preserves the dependency $C \rightarrow A$.

A direct application of the definition gives us a straightforward algoritlun for testing whether a deconlposition is dependency-preserving. (This algorithrn is exponential in the size of the dependency set. A polynomial algorithnl is available; see Exercise 19.9.)

We began this sectiol1with an exanlple of a lossless-join deC0111position that was not dependency-preserving. Other decorupositions are dependency-preserving, but not lossless. A silnple example consists of a relation ABC' with FD $A \rightarrow B$ that is decornposed into AB and BG.

19.6 NORMALIZATION

Having covered the concepts needed to understand the role of HortHa} forms and decolnpositions in database design, we now consider algoritlnIls for converting relations to BCNF or 3NF. If a relation schema is not in BCNF, it is possible to obtain a lossless-join deccunpositioll into a collection of BCNF relation schemas. Unfortunately, there may be no dependenc,y-preserving decOlnposition into a collection of BCN.F relation schernas. However, there is always a dependency-preserving, lossless-join decoruposition into a collection of 3NF relation schernas.

19.6.1 Decomposition into BCNF

We now present an algorithm for decornposing a relation scherna R with a set of FI)sF into a collection of BCNF relation schernas:

1. Suppose that R is not in BCNF. Let $X \subset R$, A be a single attribute in R, and $X \rightarrow A$ be an FD that causes a violation of BCNF. DecornposeR into R - A and XA.
2. If either R - A or XA is not in BCN.F, decornpose thern further by a recursive application of this algorithrn.

$R - A$ denotes the set of attributes other than A in R, and XA denotes the union of attributes in X and A. Since $X \rightarrow A$ violates BCNF, it is not a trivial dependency; further, A is a single attribute. Therefore, A is not in X; that is, X n A is ernpty. Therefore, each dec()lnposition carried out in Step 1 is lossless-j oin.

The set of dependencies associated with $R - A$ and XA is the projection of F onto their attributes. If one of the new relations is not in BCNF, w'e decornpose it further in Step 2. Since a decornposition results in relations with strictly fewer attributes, this process terrninates, leaving us with a collection of relation schernas that are all in BCNF. Further, joining instances of the (two or 1nore) relations obtained through this algorithrn yields precisely the corresponding instance of the original relation (i.e., the decorllposition into a collection of relations each of which in BCNF is a lossless-join dec()lnposition).

Consider the Contracts relation with attributes *C3JDPQV* and key *C.* We are given FDs $JP \rightarrow C$ and $3D \rightarrow P$. By using the dependency $3D \rightarrow P$ to guide the decornposition, we get the two schernas *3DP* and *C5JDQV. 51)P* is in BCNF. Suppose that we also have the constraint that each project deals with a single supplier: $.I \rightarrow 5$. This rneans that the sche1na *CSJDQV* is not in BCNF So we deccnnpose it further into *J3* and *C.IDC2V.* $C \rightarrow JDQV$ holds over *CJDQV;* the only other FI)s that hold are those obtained frorll this PI) by augrnentation, and therefore all FDs contain a key in the left side. Thus, each of the schernas *ST)P, .IS,* and *CJDQV* is in BCNF, and this collection of schcrnas also represents a lossless-join decornposition of *CSJDQV.*

The steps in this deC(nllposition process can be visualized as a tree, as shown in Figure 19.10. The root is the original relation *CSJIJPQV*, and the leaves are the BCNF relations that result frorn the deccHnposition aJgorithrn: *3D?, .IS,* and *CSDQV.* Intuitively, ea.ch internal node is replaced by its children through a single decomposition step guided by the FD shown just below the node.

Redundancy in BCNF Revisited

The decolnposition of *CSJDQV* iuto *SDP*, *JS*, and *CJDQV* is not dependency-preserving. Intuitively, dependency $Jp \rightarrow C$ carlllot be enforced without a, join. ()ne way to deal \vith this situation is to add a relation \vith attributes *GJ}).* In

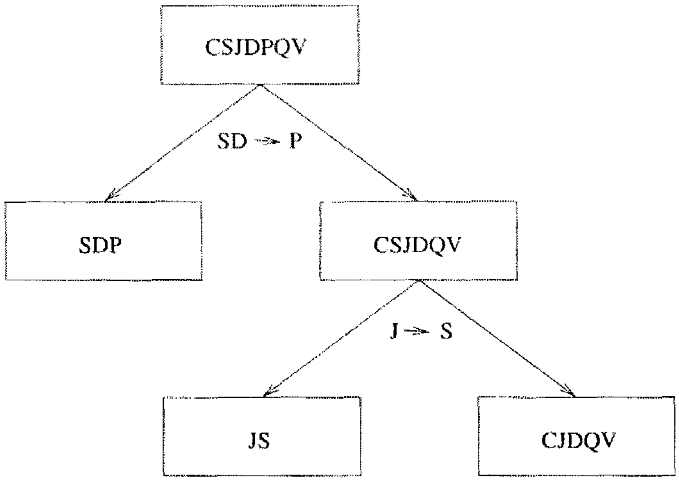

Figure 19.10 Decomposition of *CSJDQV* into *SDP*, *JS*, and *CJDQV*

effect, this solution arnounts to storing SOITle information redundantly to rnake the dependency enforcement cheaper.

This is a subtle point: Each of the schemas *CJP*, *SDP*, *JS*, and *CJDQV* is in BCNF, yet some redundancy can be predicted by FD infonnation. In particular, if we join the relation instances for *SDP* and *CJDQV* and project the result onto the attributes *CJP*, we rnust get exactly the instance stored in the relation with scherna *CJP*. We saw in Section 19.4.1 that there is no such redundancy within a single BCNF relation. This exarnple shows that redundancy can still occur across relations, even though there is no redundancy within a relation.

Alternatives in Decomposing to BCNF

Suppose several dependencies violate BCNF. Depending on ·which of these dependencies we choose to guide the next decornposition step, we rnay arrive at quite different collections of BeNF relations. Consider Contracts. We just decornposed it into *SDP*, *is*, and *CJDQV*. Suppose we choose to decornpose the original relation *CSJDPQV* into *JS* and *CJDPQV*, based on the FD $.I \rightarrow S$. The only dependencies that hold over *CJDPQV* are $.IP \rightarrow$ C and the key dependency $C \rightarrow C.IDPQV$. Since *iP* is a key, *CJDPQV* is in BeNF. Thus, the schernas *JS* and *CJDPQV* represent a lossless-join decornposition of Contracts into BCNF relations.

The lesson to be learned here is that the theor,Y of dependencies can tell us ·when there is redundancy and give us clues about possible clecornpositions to address the problern, but it cannot discrirninate arnong decornposition alternatives. A designer has to consider the alternatives and choose one based on the scrnantics of the application.

BCNF and Dependency-Preservation

Sometimes, there siIuply is no decomposition into BCNF that is dependency-preserving. As an exaruple, consider the relation schelna *SBD,* in which a tuple denotes that sailor S has reserved boat ,8 OIl date [J. If we have the FDs $8B \rightarrow D$ (a sailor can reserve a given boat for at nlost one day) and $D \rightarrow B$ (on any given day at rllost one boat can be reserved), *SBn* is not in BCNF because D is not a key. If we try to dec(nnpose it, however, we cannot preserve the dependency $BB \rightarrow D$.

19.6.2 Decomposition into 3NF

Clearly, the approach we outlined for 10ssless-joiIl decornpositioIl into BCNF also gives us a lossless-join decomposition into 3NF. (Typically, we can stop a little earlier if we are satisfied with a collection of 3NF relations.) But this approach does not ensure dependency-preservation.

A siInple rllodification, however, yields a deco111position into 3NF relations that is lossless-join and dependency-preserving. Before we describe this modification, we need to introduce the concept of a lninirnal cover for a set of FDs.

Minimal Cover for a Set of FDs

A minimal cover for a set F of FDs is a set G of FDs such that:

1. Every dependency in G is of the forIn $X \rightarrow A$, where A is a single attribute.
2. The closure $F+$ is equal to the closure (;+.
3. If we obtain a set *II* of dependencies frorn G by deleting one or 1110re dependencies or by deleting attributes frorn a dependency in G, then $p+' \neq II+$.

Intuitively, a rninirnal cover for a set F of FDs is an equivalent set of dependencies that is *minimal* in two respects: (1) Every dependency is as sIIall as possible; tha,t; is, each attribute on the left side is necessary and the right side is a single attribute. (2) Every dependency in it is required for the closure to be equal to F^+.

As an exarnplc, let F be the set of dependencies:

$$it \rightarrow B,\ ABCID \rightarrow E,\ EF \rightarrow G,\ EF \rightarrow II,\ \text{and}\ ACDF \rightarrow EG.$$

First, let us rewrite $itCDF \rightarrow BG$ so that every right side is a single attribute:

$ACDF \to E$ and $ACDF \to G$,

Next consider $ACDF \to G$, This dependency is irnplied by the following FDs:

$A \to B$, $ABCD \to E$, and $EF \to G$,

Therefore, we can delete it, Sirnilarly, we can delete $ACDF \to E$. Next consider $ABCD \to E$, Since $A \to B$ holds, we can replace it with $ACD \to E$, (At this point, the reader should verify that each rernaining FD is rninilnal and required,) Thus, a minimal cover for F is the set:

$A \to B$, $ACD \to E$, $EF \to G$, and $EF \to H$,

The preceding exarnple illustrates a general algorithrn for obtaining a rninimal cover of a set F of FDs:

1. Put the FDs in a Standard Form: Obtain a collection G of equivalent FDs with a single attribute on the right side (using the decornposition axiolIl),
2. Minimize the Left Side of Each FD: For each FD in G, check each attribute in the left side to see if it can be deleted while preserving equivalence to $F+$,
3. Delete Redundant FDs: Check each reluaining FD in G to see if it can be deleted while preserving equivalence to $F+$,

Note that the order in which we consider FDs while applying these steps could produce different rninilnal covers; there could be several rninirnal covers for a given set of FDs,

IV101'8 irnportant, it is necessary to rniniInize the left sides of F'Ds *before* checking for redundant FI)s, If these two steps are reversed, the final set of FI)s could still contain senne redundant FDs (i,e., not be a rninirnal cover), as the following exarnple illustrates, LetF be the set of dependencies, each of which is already in the standard fornl:

$ABCD \to E$, $E \to D$, $A \to B$, and $AC \to D$,

Observe that none of these FDs is redundant; if we checked for redundantFDs first, we would get the same set of FI)s F. The left side of *il13CIJ* E can be replaced by AC while preserving equivalence to F^+, and we \vould stop here if \ve checked for reclunda.ntF'Ds in F before rnillilnizing the left sides. However, the set of FDs we have is not a Ininimal cover:

$AC \rightarrow E, E \rightarrow D, A \rightarrow B$, and $AC \rightarrow D$.

From transitivity, the first two FDs irnply the last FD, which can therefore be deleted while preserving equivalence to F^+. The irnportant point to note is that $AC \rightarrow D$ becc)lnes redundant only after we replace $ABeD \rightarrow E$ with $AC \rightarrow E$. If we Ininirnize left sides of FDs first and then cheek for redundantFDs, we are left with the first three FDs in the preeeding list,whieh is indeed a Ininirna1 cover for F.

Dependency-Preserving Decomposition into 3NF

Returning to the problenl of obtaining a lossless-join, dependency-preserving decornposition into 3NF relations, let R be a relation with a set [/' of FDs that is a minirnal cover, and let $R_1, R_2, \ldots, R_n$ be a lossless-join decolnposition of R. For $1 \leq i \leq n$, suppose that each R_i is in 3NF and let F_i denote the projection of F onto the attributes of R_i. Do the following:

- Identify the set N of dependencies in F that is not preserved, that is, not included in the closure of the union of *Fis*.
- :F'or each FD $X \rightarrow A$ in N, create a relation schelna XA and add it to the decomposition of R.

Obviously, every dependency in F is preserved if we replace R by the R_is plus the schernas of the fornl XA added in this step. The *Ris* are given to be in 3NF. We can show that each of the schemas XA is in 3NF as follows: Since $X \rightarrow A$ is in the lninirnal cover F, $Y \rightarrow A$ does not hold for any Y that is a strict subset of X. Therefore, X is a key for XA. :F\llrther, if any other dependencies hold over XA, the right side can involve only attributes in X' because A is a single attribute (because $X \rightarrow A$ is an FD in a rninhnal cover). Since X is a key for ..:YA, none of these additional dependencies causes a violation of 3NF (although they rnight cause a violation of BCNF).

As an optilYlization, if the set N contains several FI)swith the saIne left side, say, $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$, we can replace thern \vith a single equivalent FD $X \rightarrow$ AI $\ldots A_n$. Therefore, we produce one relation scherna $XA_1 \ldots A_n$, instead of several schernas $XA_1, \ldots, XA_n$, \vhich is generally preferable.

Consider the Contracts relation with attrilnltes $CSJDPQV$ and FI)s $JP \rightarrow C$, $SD \rightarrow P$, and $J \rightarrow S$. If we decolnpose $CSJDPQV$ into *SDIJ* and $CSJDQV$, then $8DP$ is in BCNF, but $CSJDQV$ is not even in 3NF. So \ve dec.olupose it further into JS and $CJDQV$. rrhe relation schemas SDP, *.I8*, and $CJDQV$ are in 3NF (in fact, in BCNF), and the decoInposition is lossless-join. However, the dependency $JP \rightarrow C$ is not preserved. This problerIl can be addressed by adding a relation schema CJP to the decornposition.

3NF Synthesis

We assurned that the design process starts with an ER diagraII1, and that our use of FDs is primarily to guide decisions about decolnposition. The algorithill for obtaining a lossless-join, dependency-preserving decornpositiol1 was presented in the previous section fro111 this perspective--------a lossless-join decoru-position into 3NF is straightforward, and the algorithrn addresses dependency-preservation by adding extra relation schcrnas.

An alternative approach, called **synthesis**, is to take all the attributes over the original relation R and a rnininlal cover F for the FDs that hold over it and add a relation scherna XA to the decomposition of R for each FD $X \rightarrow A$ in F.

The resulting collection of relation schernas is in 3NF and preserves all FDs. If it is not a lossless-join decomposition of R, we can Dlake it so by adding a relation schenla that contains just those attributes that appear in sorne key. This algorithrn gives us a lossless-join, dependency-preserving decornposition into 3NF and has polynornial corIlplexity-----polynornial algorithms are available for coruputing rninirnal covers, and a key can be found in polync)Inial tirHe (even though finding all keys is known to be NP-cornplete). The existence of a polynornial algorithnl for obtaining a lossless-join, dependency-preserving decornposition into 3NF is surprising when we consider that testing whether a given schema is in 3NF is NP-cornplete.

As an exarnple, consider a relation ABC with FI)s $F = \{A \rightarrow B, C \rightarrow B\}$. The first step yields the relation scheluas AB and BG. This is not a lossless-join deC0l11position of $AilC$; $AB \cap BC$ is B, and neither $B \rightarrow A$ nor $B \rightarrow C$ is in F^+. If we add a schema AC, we have the lossless-join property as well. Although the collectic)ll of relations AB, BC, and AC is a depenclency-preserving, lossless-join decornposition of ABC, we obtained it through a process of *synthesis*, rather tllan through a process of repeated decornposition. We note that the decoIIIposition produced by the synthesis approa,ch heavily dependends on the rninirnal cover used.

As another example of the synthesis approach, consider the Contracts relation with attributes $CSJDPQV$ and the follovving FI)s:

$$C \quad CSJDPQV, .IP \rightarrow C, 8D \rightarrow P, \text{ and } J \rightarrow S.$$

This set of FI)s is not a rninirnal cover, and so we IIIUSt find OI18. We first replace $G \rightarrow CSJDPQV$ with tllcF'I)s:

$C \rightarrow 5$, $C \rightarrow J$, $C \rightarrow IJ$, $C \rightarrow P$, $C \rightarrow Q$, and $C \rightarrow V$.

The FD $C \rightarrow P$ is irnplied by $C \rightarrow S$, $C \rightarrow D$, and $SD \rightarrow P$; so we can delete it. The FD $C \rightarrow S$ is irnplied byC $\rightarrow J$ and $J \rightarrow S$; so we can delete it. This leaves us with a rninirnal cover:

C $\rightarrow$,*I*, C $\rightarrow$ 1), $C \rightarrow Q$, C $\rightarrow V$, $JP \rightarrow$ C, $3D \rightarrow P$, and $J \rightarrow S$.

lJsing the algorithrll for ensuring dependency-preservation, we obtain the relational scherna *CJ, CD, CQ, CV, GIJP, SDP,* and *JB*. We can irnprove this schenla by cornbining relations for which C is the key into *CDJPQV*. In addition, we have *SDP* and *,IS* in our decorllposition. Since one of these relations (*CDJPQ*V) is a superkey, we are done.

Conlparing this decomposition with that obtained earlier in this section, we find they are quite close, with the only difference being that one of them has *CDJPQV* instead of *CJP* and *CJDQV*. In general, however, there could be significant differences.

19.7 SCHEMA REFINEMENT IN DATABASE DESIGN

We have seen how normalization can eliminate redundancy and discussed several approaches to nonnalizing a relation. We now consider how these ideas are applied in practice.

Database designers typically use a conceptual design rnethodology, such as ER design, to arrive at an initial database design. Given this, the approach of repeated decorllpositions to rectify instances of redundancy is likely to be the rnost natural use of PI)s and nonnalization techniques.

In this section, we Inotivate the need for a schcrna refinernent step follovving ER design. It is natural to ask whether we even need to decornpose relations produced by translating an ER diagranl. Should a good ER design not lead to a collection of relations free of redundancy prob.lerns? Unfortunately, ER design is a c()!nplex, subjective process, and certain constraints are not expressible in tenns of ER diagraJns. The exaruples in this section are intendecl to illustrate why decornposition of relations produced through ER design rnight be necessary.

19.7.1 Constraints on an Entity Set

Consider the Hourly_Emps relation again. rrhe constraint that attribute *ssn* is a key can be expressed as an FI):

$$\{ssn\} \rightarrow \{ssn,\ name,\ lot,\ rating,\ hourly\_wages,\ hours\_worked\}$$

:For brevity, we \vrite this FD as $S \rightarrow SNLRWH$, using a single letter to denote each attribute and ornitting the set braces, but the reader should rernernber that both sides of an FD contain sets of attributes. In addition, the constraint that the *hourly_wages* attribute is deterruined by the *rating* attribute is an FD: $R \rightarrow W$.

As we saw in Section 19.1.1, this FI) led to redundant storage of rating wage associations. *It cannot be expressed in terms of the ER model. Only FDs that determine all attributes of a relation (i.e., key constraints) can be expressed in the ER rnodel.* rrherefore, we could not detect it when we considered Hourly_EIIlPS as an entity set during ER IIlodeling.

We could argue that the problenl with the original design was an artifact of a poor ER design, which could have been avoided by introducing an entity set called Wage_Table (with attributes *rating* and *hourly_wages*) and a relationship set lIas_Wages associating IIourly_.ErIlps and Wage_Table. The point, however, is that we could easily arrive at the original design given the subjective nature of ER rnodeling. Having forInal techniques to identify the problenl with this design and guide us to a, better design is very useful. The value of such techniques cannot be underestirnated when designing large schernas·····-schcrnas with rnore than a hundred tables are not unCOIllIHon.

19.7.2 Constraints on a Relationship Set

The previous exarnple illustrated how FDs can help to refine the subjective decisions Blade during ER. design, but one could argue that the best possible ER, eliagrarn \vould have led to the same final set of relations. ()ur next exarnple shows how Ff) inforrnation ca.ll lead to a. set of relations unlikely to be arrived at solely through ER design.

We revisit an example frOTH Chapter 2. Suppose that we have entity sets Parts, Suppliers, and I)epartrnents, as \vell as a relationship set Contracts that involves all of theIn. We refer to the scherna for Contra(:ts as *CQPSD*. A contra,ct with contract id C specifies that a supplier S will supply sorne quantity Q of a part P to a departrnent J). (We have adderl the contract ieI field C' tC) tIle versiorl of the Contracts relation discussed in Chapter 2.)

We Blight have a policy that a departrnent purchases at Inost one part frorl1 any given supplier. Therefore, if there are several contracts between the saIne supplier and departrnent, \ve know that the saIne part Inus!; be involved in all of thcrn. This constraint is an FD, $DS \rightarrow P$.

Again we have redundancy and its associated problclns. We can address this situation by decornposing Contracts into two relations with attributes *CQSD* and *3DP*. Intuitively, the relation *3DP* records the part supplied to a depart-rllent by a supplier, and the relation *C:QSD* records additional infornlation about a contract. It is unlikely that we would arrive at such a design solely through ER rIlodeling, sinee it is hard to fOfrnulate an entity or relationship that corresponds naturally to *CQSD*.

19.7.3 Identifying Attributes of Entities

This exarIlple 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 diagrarn in Figure 19.11 shows a relationship set called Works_In that is similar to the Works.ln relationship set of Chapter 2 but with an additional key constraint indicating that an employee can work in at rnost one departrIlent. (Observe the arrow connecting Employees to Works_In.)

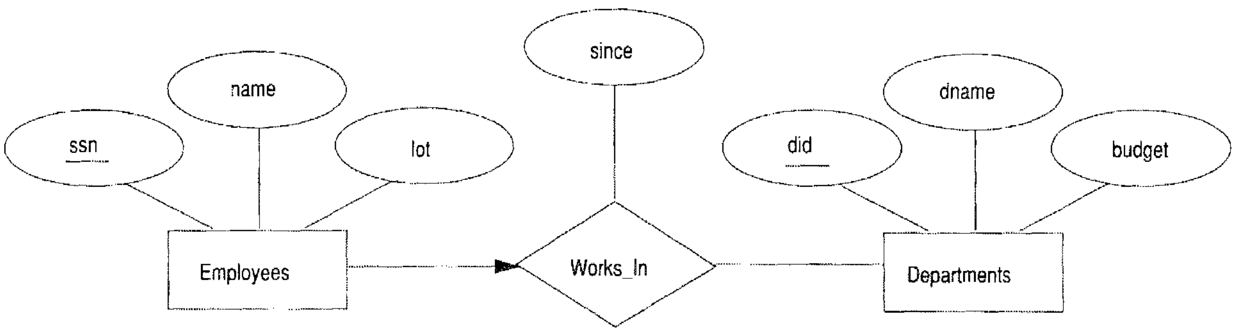

Figure 19.11. The Works._In Relationship Set

Using the key constraint, we can translate this ER diagrarn into two relations:

Workers(*ssn*, *narne*, *lot*, *d'id*, *since*)
Departrnents(*did*, *dname*, *budget*)

The entity set Ernployees and the relationship set Works_In are rnapped to a single relation, vVorkers. This translation is based on the second approach discussed in Section 2.4.1.

Now suppose elIlployees are assigned parking lots based on their departrnent, and that all enlployees in a given departrnent are assigned to the saIne lot. This constraint is not expressible with respect to the ER, diagrarIl of Figure 19.1l. It is another example of an FD: *did* → *lot.* The redundancy in this design can be elirninated by decornposing the Workers relation into two relations:

vVorkers2(*ssn*, *name*, *did*, *since)*
Dept_Lots(*did*, *lot)*

'rhe new design has lnuch to reconunend it. We can change the lots associated with a departlnent by updating a single tuple in the second relation (i.e., no update anornalies). We can associate a lot with a department even if it currently has no crnployees, without using *null* values (i.e., no deletion anornalies). We can add an eruployee to a department by inserting a tuple to the first relation even if there is no lot associated with the enlployee's departrnent (i.e., no insertion anornalies).

Exalnining the two relations Departrnents and Dept_Lots, which have the saIne key, we realize that a Departrnents tuple and a Dept_Lots tuple with the sarne key value describe the sarne entity. This observation is reflected in the ER cliagrarn shown in Figure 19.12.

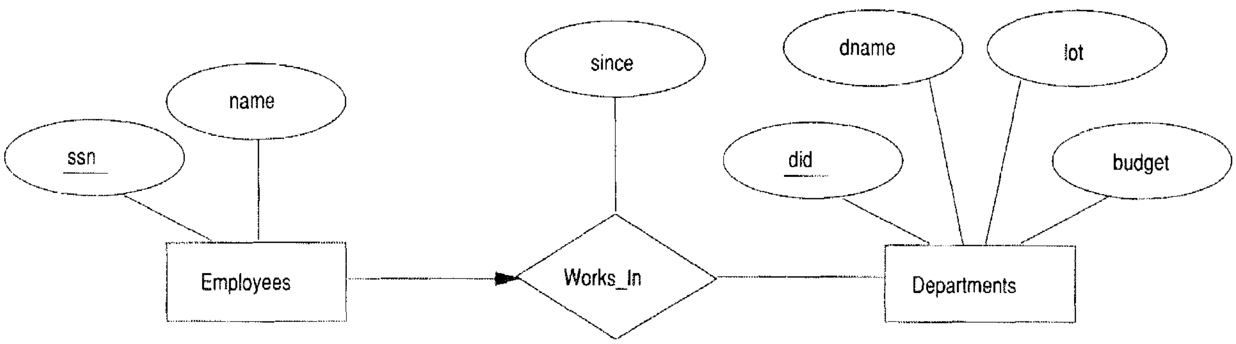

Figure 19.12 Refined\Norks_In Relationship Set

Translating this diagrarn into the relational rnodel would yield:

Workers2(8871" *narne*, *did*, *since)*
I)epartrnentsCdid, *dname*, *budget*, *lot)*

It SeClllS intuitive to associate lots with crnployees; on the other hand, the les reveal tllat ,in this exarnple lots are really associated with departrnents. The subjective process of ER modeling could Iniss this point. T'he rigorous process of norrnaliza,tion would not.

19.7.4 Identifying Entity Sets

Consider a variant of the Reserves scherna used in earlier chapters. 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 C denoting the credit card to which the reservation is charged. We use this exarnple to illustrate how FD illfonnation can be used to refine an ER design. In particular, we discuss how FD inforluation can help decide whether a concept should be rnodeled as an entity or as an attribute.

Suppose every sailor uses a unique credit card for reservations. This constraint is expressed by the FD $S \rightarrow C$. This constraint indicates that, in relation Reserves, we store the credit card rnllnber for a sailor as often as we have reservations for that sailor, and we have redundancy and potential update anolnalies. A solution is to deconlpose Reserves into two relations with attributes SBD and SC. Intuitively, one holds infornation about reservations, and the other holds infonnation 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 Inap Has_Card and Credit_Cards to a single relation with attributes SC. We would probably not rnodel credit card nUlnbers as entities if our Inain interest in card nurnbers is to indicate how a reservation is to be paid for; it suffices to use an attribute to rnodel card nUlnbers in this situation.

A second approach is to rnake *cardno* an attribute of Sailors. But this approach is not very natural-.–a sailor lllay have several cards, and we are not interested in all of theln. Our interest is in the one card that is used to pay for reservations, which is best lnodeled as an attribute of the relationship Reserves.

A helpful way to think about the design problern in this exarnple is that we first lnake *cardno* an attribute of H,eserves and then refine the resulting tables by taking into account the FD information. (Whether we refine the design by adding *cardno* to the table obtained froTll Sailors or by creating a new table with attributes SC is (1, separate issue.)

19.8 OTHER KINDS OF DEPENDENCIES

FI)s are probal)l.y the rn08t conunon and irnportant kind of constraint froIII the point of view of database design. However, there are several other kinds of dependencies. In particular, there is a well-developed theory for database design llsing *multivalued dependencies* and *join dependencies*. By taking sueh dependencies into account, we can identify potential redundancy problenls that cannot be detected using FDs alone.

'rhis section illustrates the kinds of redundancy that can be detected using II1Ul-tivalued dependencies. Our Inain observation, however, is that sirnple guidelines (which can be checked using only FD reasoning) can tell us whether we even need to worry about complex constraints such as 111ultivalued and join dependencies. We also conunent on the role of *inclusion dependencies* in database design.

19.8.1 Multivalued Dependencies

Suppose that we have a relation with attributes *course, teacher,* and *book,* which we denote as *CTB*. The Ineaning 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 recolnlnended 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 R.elation with Redundancy That Is Revealed by MVDs

Note three points here:

- The relation sehcrna *CTB* is in BCNF; therefore we would not consider decolnposing it further if we looked only at the FDs that hold over *(JTB.*
- There is redundancy. rrhe fact that G·reen can teach Physics101 is recorded once per recommended text for the course. Sirnila.rly, the fact that Optics is a text for Physics101 is recorded once per potential teacher.
- T'he redundancy can be elirninated by decornposing *CTB* into CT and *CE.*

The redundaJ1cy in this exarnple is due to the constraint that the texts for a course are independent of tIle instructors, which cannot be expressed in tenns of FDs. rrhis constraint is an example of a *multivalued dependency,* or MVD. Ideally, we should rnodel this situation using two binary relationship sets, Instructors with attributes CT and Text with attributes CB. Because these are two essentially independent relationships, rnodeling them with a single ternary relationship set with attributes CTE is inappropriate. (See Section 2.5.3 for a further discussion of ternary versus binary relationships.) Given the subjectivity of ER design, ho\vever, we rnight create a ternary relationship. A careful analysis of the fvIVD infonnation would then reveal the problern.

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

ForInally, if the MVD $X \rightarrow\rightarrow Y$ holds over R and $Z = R - XY$, the following lllUSt be true for every legal instance r of R:

> If tl E r, $t2$ E rand $tl.X = t2.X$, then there must be sorne $t3$ E r such that $tl'XY = t3.XY$ and $t2 \cdot Z = t3'Z$,

Figure 19.14 illustrates this definition. If we are given the first two tuples and told that the MVD $X \rightarrow\rightarrow Y$ 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 t_1—we can deduce that the tuple $t4$ must also be in the relation instance.

LX	Y	Z]	
a	b_1	CI	— tuple $t1$
a	b_2	C2	— tuple $t2$
a	bl	C2	— tuple t_3
a	$b2$	CI	— tuple $t4$

Figure 19.14 Illustration of MVD Definition

This table suggests another way to think about lVIVDs: If $X \rightarrow\rightarrow Y$ holds over R, then $\pi_{YZ}(\sigma_{X=x}(R)) = \pi_Y(\sigma_{X=x}(R)) \times \pi_Z(\sigma_{X=x}(R))$ in every legal instance of R, for any value x that appears in the X colurnn of R. In other words, consider groups of tuples inR with the sarne X-value. In each such group consider the projection onto the attributes YZ. This projection HUlst be equal to the cross-product of the projectiolls onto Yand Z. That is, for a given X-value, the Y-values and Z-values are independent. (Froln this definition it is easy to see that $X \rightarrow\rightarrow$ y'l11Ust hold wherlever $X \rightarrow Y$ holds. If the FI) $X \rightarrow$

Y holds, there is exactly one Y-value for a given X-value, and the conditions in the MVD definition hold trivially. The converse does not hold, as Figure 19.14 illustrates.)

Returning to our CTB exaIllple, the constraint that course texts are independent of instructors can be expressed as $C \rightarrow\rightarrow T$. In terlllS of the definition of MVDs, this constraint can be read as follo\vs:

> If (there is a tuple showing that) C is taught by teacher T,
> and (there is a tuple showing that) G has book B as text,
> then (there is a tuple showing that) G is taught by T and has text B.

Given a set of FDs and MVDs, in general, we can infer that several additional FDs and MVDs hold. A sound and complete set of inference rules consists of the three ArIllstrong AxioIllS plus five additional rules. Three of the additional rules involve only MVDs:

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

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

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

()bserve that replication states that every FD is also an MVD.

19.8.2 Fourth Normal Form

Fourth Horrnal fonn is a direct generalization of BeNF. Let R be a relation scherna, X and Y be nonernpty 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 l\1.VI) $X \rightarrow\rightarrow Y$ that holds over R, one of the following staternents is true:

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

In reading this definition, it is irnportant to understand that the deflnition of a *key* has not changed··········-the key rnust uniquely deterrnine all attributes through FDs alone. $X \rightarrow\rightarrow$ Y'is a trivial MVD if $Y \subset X \subseteq R$ or $XY = R$; such MVDs always hold.

The relation *CTB* is not in 4NF because $C \rightarrow\rightarrow T$ is a nontrivial MVD and C is not a key. We can elirninate the resulting redundancy by deconlposing *CTB* into *CT* and *CB;* each of these relations is then in 4NF.

To use MVD inforrnation fully, we nUlst 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 inforrnation. That is, 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 in1.portant assl.unption is inlplicit 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 exalnple. Consider a relation scherna *ABCD* and suppose that the FD $A \rightarrow BCD$ and the MVD $B \rightarrow\rightarrow C$ are given. Considering only these dependencies, this relation schema appears to be a counterexalnple to the result. The relation has a sirnple key, appears to be in BCNF, and yet is not in 4NF because $B \rightarrow\rightarrow C$ causes a violation of the 4NF conditions. Let us take a closer look.

B	C	A	D	
b	c_1	a_1	d_1	— tuple t_1
b	c_2	a_2	d_2	— tuple t_2
b	c_1	a_2	d_2	— tuple t_3

Figure 19.15 Three Tuples [rorn a Legal Instance of *ABCD*

Figure 19.15 8ho\v8 three tuples f1'om an instance of *ABCD* that satisfies the given MVD $B \rightarrow\rightarrow C$. Frolu the definition of an MVD, given tuples tl and t_2, it follows that tuple t_3 Inust also be included in the instaJ1ce. Consider tuples t_2 and t_3. FroIJI the givenFD $A \rightarrow BCD$ and the fact that these tuples have the same A-value, we can deduce that $c_1 = c_2$. Therefore, we see that the FD $B \rightarrow C$ rnust hold over $ABCD$ whenever the FD $A \rightarrow BCD$ and the MVD $B \rightarrow\rightarrow C$ hold. If $B \rightarrow C$ holds, the relation $ABeD$ is not in BeNF (unless additional FDs lllake B a key)!

Thus, the apparent counterexalnple is really not a counterexalllple···········-rather, it illustrates the iInportance of correctly identifying all FDs that hold over a relation. In this exarnple, $A \rightarrow BCI)$ is not the only FD; the FD $B \rightarrow C$ also holds but ·was not identified initially. Given a set of FDs and IvIVI)s, the inference rules can be used to infer additional FDs (and l\1VDs); to apply the Date-Fagin result without first using the l\1VD inference rules, we IUUSt be certain that we have identified all the FDs.

In summary, the Date-Fagin result offers a convenient way to check that a relation is in 4NF (without reasoning about l\1VDs) if we are confident that we have identified all FDs. At this point, the reader is invited to go over the examples we have discussed in this chapter and see if there is a relation that is not in 4NF.

19.8.3 Join Dependencies

A join dependency is a further generalization of MVDs. A join dependency (JD) $\bowtie \{R_1, \ldots, R_n\}$ is said to hold over a relation R if $R_1, \ldots, R_n$ is a lossless-join decolnposition of R.

An MVD $X \rightarrow\rightarrow Y$over a relation R can be expressed as the join dependency $\bowtie$ {XV, X(R,--Y)}..As an example, in the GTB relation, the MVD $C \rightarrow\rightarrow T$ can be expressed as the join dependency $\bowtie$ {Crr, CB}.

U·nlike FDs and l'v1VDs, there is no set of sound and cornplete inference rules for JDs.

19.8.4 Fifth Normal Form

A relation schcrna R is said to be in fifth normal form (5NF) if, for every .II) $\bowtie \{R_1, \ldots, R_n\}$ that holds over R, one of the follo"ving statcrnents is true:

- $R_i = R$, for scnne i, or
- The .lD is irnplied by the set of those FDs over R in ·which the left side is a key for R.

The second condition deserves s(Hne explanation, since we have not presented inference rules for FDs and .00Ds taken together. Intuitively, we rnust be able to sho\v that the decolnposition of R into $\{R_1, \ldots, R_n\}$ is lossless-join whenever the key dependencies (FDs in which the left side is a key for R) hold. JI) $\bowtie \{R_1, \ldots, R_n\}$ is a trivial JD if $R_i = R$ for SaIne i; such a JD always holds.

The following result, also due to Date and Fagin, identifies conditions—again, detected llsing only FD inforlnation---under -which we can safely ignore JD inforlnation:

> If a relation schenla is in 3NF and each of its keys consists of a single attribute, it is also in 5NF.

The conditions identified in this result are sufficient for a relation to be in 5NF but not necessary. rrhe result can be very useful in practice because it allows us to conclude that a relation is in 5NF *'Without ever 'identifying the MVDs and JDs that 'may hold oveT the relation.*

19.8.5 Inclusion Dependencies

lVIVDs and JDs can be used to guide database design, as we have seen, although they are less COllUllon than FDs and harder to recognize and reason about. In contrast, inclusion dependencies are very intuitive and quite cornrnon. IIowever, they typically have little influence on database design (beyond the ER design stage).

Infonnally, an inclusion dependency is a statement of the fOlTH that s0111e cohunns of a relation are contained in other cohunns (usually of a second relation). A foreign key constraint is an example of an inclusion dependency; the referring colurnn(s) in one relation rnust be contain.ecl in the prirnary key cohnnn(s) of the referenced relation. As another exarnple, if!? and S are two relations obtained 1)y translating two entity sets that every R entity is also an S erltity, we would have an inclusion dependency; projecting R on its key attributes yields a relation contained in the relation obtained by projecting S on its key attributes.

The rnain point to bear in rnind is that we should not split groups of attributes that participate in an inclusion dependency. For exarnple, if we have an inclusion dependency $AB \subseteq Of)$, \vhile decornposing the relation scherna containing AB, we should ensure that at least one of the schemas obtained in the deccnnposition contains botJ1 A and B. Otherwise, we cannot check the inclusion clependency $AB \subseteq CD$ without reconstructing the relation containing $A13$.

Ivlost inclusion dependencies in practice are *key-based*, that is, involve only keys. Foreign key constraints are a good exalnple of key-based inclusion dependencies. An ER diagram that involves ISA hierarchies (see Section 2.4.4) also leads to key-based inclusion dependencies. If all inclusion dependencies are key-based, we rarely have to worry about splitting attribute gTOUps that participate in inclusion dependencies, since decornpositions usually do not split the primary key. N'ote, however, that going fn:)111 3NF to BCNF always involves splitting SOlne key (ideally not the prirnary key!), since the dependency guiding the split is of the fornl $X \rightarrow A$ where A is part of a key.

19.9 CASE STUDY: THE INTERN'ET SHOP

R,ecall froIn Section 3.8 that DBDudes settled on the following scherna:

> Books(*isbn:* CHAR(10), *title:* CHAR(8), *author:* CHAR(80),
> *qty_in_stock:* INTEGER, *price:* REAL, *year_published:* INTEGER)
> Custolllers(*cid:* INTEGER, *cnaTne:* CHAR(80), *address:* CHAR(200))
> Orders(*orde.rnum,:* INTEGER, *isbn:* CHAR(.10), *cid:* INTEGER,
> *cardnu'm:* CHAR(16), *qty:* INTEGER, *ordeT_date:* DATE, *ship_date:* DATE)

DBDudes analyzes the set of relations for possible redundancy. The Books relation has only one key, *(isbn),* and no other functional dependencies hold over the table. Thus, Books is in BCNF. The Custorners relation also has only one key, (*cid*), and no other functional depedencies hold over the table. T'hus, Custorners is also in BCNF.

DBI)udes has already identified the pair $\langle ordernum, isbn \rangle$ as the key for the Orders table. In addition, since each order is placed by one custorner on one specific date with one specific credit card nurnber, the following three functional dependencies hold:

$$ordernum \rightarrow cid,\ ordernum \rightarrow order\_date, \text{ and } ordernum \rightarrow cardnum$$

The experts at DBDudes conclude that Orders is not even in 3NF. (Can you see why?) They decide to clecornpose ()rders into the following two relations:

> Orders(*ordernum*, *cid*, *order_date*, *cardnum*, a.nd
> ()rderlists(*ordernum*, *isbn*, *qty*, *ship_date*)

The resulting t\vo relations, ()rders and ()rderlists, are both in BCNF', and the decornposition is lossless-join since *ordernum* is a key for (the new) ()rders. The reader is invited to check that this decolnposition is also dependency-preserving. For cornpleteness, we give th,e SQL DIJL for the ()rders and Orderlists relations below:

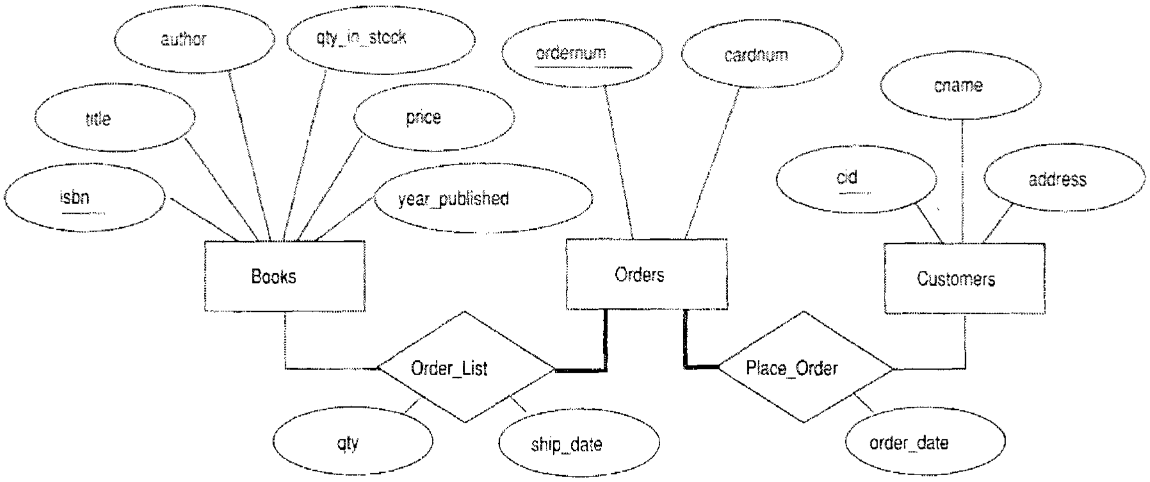

Figure 19.16 ER Diagram Reflecting the Final Design

```
CREATE TABLE Orders ( ordernurn      INTEGER,
                      cid            INTEGER,
                      order_date     DATE,
                      cardnum        CHAR(16),
                      PRIMARY KEY (ordernlllll),
                      FOREIGN KEY (cid) REFERENCES Custolllers )

CREATE TABLE Orderlists (ordernurll    INTEGER,
                         isbn          CHAR(10),
                         qty           INTEGER,
                         ship_date     DATE,
                         PRIMARY KEY (ordernurn, isbn),
                         FOREIGN KEY (isbn) REFERENCES Books)
```

F'igure 19.16 shows an updated ER diagrarn that reflects the new design. Note that DBDudes could have arrived inunedia,tely at this diagrarn if they had made ()rders an entity set instead of a relationship set right at the beginning. But at that tilne they did not understand the requirernents cornpletely, and it seeTHed natural to rnodel Orders as a relationship set. This iterative refinernent process is typical of real-life da,tabase design processes. As DBI)udes has learned over tirne, it is rare to achieve an initial design that is not changed as a project progresses.

T'he DBI)udes team celebrates the successful cornpletion of logical database design and scherna refinelnent by opening a bottle of charnpagne and charging it to B&:N. After recovering frorn the celebration, they lIlove on to the physical design phase.

19.10 REVIEW QUESTIONS

Answers to the review' questions can be found in the listed sections.

- Illustrate redundancy and the problerns that it ca.n cause. Give examples of *insert, delete,* and *update* anoInalies. Can *null* values help address these problerlls? Are they a cOlnplete solution? (Section 19.1.1)
- What is a *decoTnpositio'n* and how does it address redundancy? What problerlls Inay be caused by the use of decolupositions? (Sections 19.1.2 and 19.1.3)
- Define *functional dependencies.* How are *prımary keys* related to FDs? (Section 19.2)
- When is an PD *j implied by* a set *F* of FDs? Define *Armstrong's Axioms,* and explain the statement that "they are a sound and cornplete set of rules for FD inference." (Section 19.3)
- What is the *dependency closure* *F+* of a set *F* of FDs? What is the *attribute closure* *X+* of a set of attributes *X* with respect to a set of FDs *F?* (Section 19.3)
- Define INF, 2NF, 3NF, and BCNF. What is the nlotivation for putting a relation in BCNF? What is the motivation for 3NF? (Section 19.4)
- When is the decomposition of a relation schenla *R* into two relation schemas *X* and *Y* said to be a *lossless-join* decomposition? Why is this property so irnportant? Give a necessary and sufficient condition to test whether a decc)1nposition is lossless-join. (Section 19.5.1)
- When is a decornposition said to be *depc'ndency-preserving?* Why is this property useful? (Section 19.5.2)
- Describe how we can obtain **a** lossless-join decornposition of a relation into BCNF. Give an exanlple to show that there rnay not be a dependency-preserving decornposition into BCNF. Illustrate how a given relation could be decornposed in different ways to arrive at several alternative decornpositions, and discuss the irnplications for database design. (Section 19.6.1)
- Give an example that illustrates how a collection of relations ııı BCNF could have redundancy even though each relation, by itself, is free fronl redundancy. (Section 19.6.1)
- What is a. *Tninirnal cover* for a set of FDs? Describe an algorithrn for cornputing the minimal cover of a set of FI)s, and illustrate it with an exarnple. (Section 19.6.2)

- Describe how the algorithl11 for lossless-join decolnposition into BCNF can be adapted to obtain a lossless-join, dependency-preserving decornposition into 3NF. Describe the alternative *synthesis* approach to obtaining such a decorllposition into 3NF. Illustrate both approaches using an exarnple. (Section 19.6.2)
- Discuss how scherna refinernent through dependency analysis and norrnalization can iInprove schemas obtained through ER design. (Section 19.7)
- Define *multivalued dependencies, join dependencies,* and *inclusion dependencies.* Discuss the use of such dependencies for database design. Define 4NF and 5NF, and explain how they prevent certain kinds of redundancy that BCNF does not eliminate. Describe tests for 4NF and 5NF that use only FDs. What key assumption is involved in these tests? (Section 19.8)

EXERCISES

Exercise 19.1 Briefly answer the following questions:

1. Define the term *functional dependency.*
2. Why are some functional dependencies called *trivial*?
3. Give a set. of FDs for the relation schema $R(A,B,C,Dj$ with prilnary key AB under which R is in 1NF but not in 2NF.
4. Give a set of FDs for the relation schelna $R(A,B,C,Dj$ with prilnary key AB under which R is in 2NF but not in 3NF.
5. Consider the relation schelna $R(A,B,OJ$, which has the FD $B \rightarrow C$. If A is a candidate key for R, is it possible for R to be in BCNF? If so, under what conditions? If not, explain why not.
6. Suppose we have a relation schema $R(A,B,OJ$ representing a relationship between two entity sets with keys A and 13, respectively, and suppose that R has (aIllong others) the FDs $A \rightarrow$ Band 13 $\rightarrow A$. Explain what such a pair of dependencies means (i.e., what they irnply about the relationship that the relation nlOdels).

Exercise 19.2 Consider a relation R with five attributes $ABCDE$. You are given the follc)\ving dependencies: $A \rightarrow B$, $Be \rightarrow E$, and $ED \rightarrow A$.

1. List all keys for R.
2. Is R in 3NF?
3. Is R in BCNF?

Exercise 19.3 Consider the relation shown in Figure 19.17.

1. List all the functional dependencies that this relation instance satisfies.
2. Assume that the value of attribute Z of the last record in the relation is changed frorH z_3 to z_2. Now list all the functional dependencies that this relation instance satisfies.

Exercise 19.4 Assurne that you are given a relation with attributes $ABCD$.

X	Y	Z
Xl	y_1	Zl
Xl	Yl	Z2
X2	Yl	Zl
x_2	Yl	z_3

Figure 19.17 Relation for Exercise 19.3.

1. Asslune that no record has NULL values. \Nrite an SQL query that checks whether the functional dependency $A \rightarrow B$ holds.
2. Assulne again that no record has NULL values. Write an SQL assertion that enforces the functional dependency $A \rightarrow B$.
3. Let us now aSSUlne that records could have NULL values. Repeat the previous two questions under this assurnption.

Exercise 19.5 Consider the following collection of relations and dependencies. Assume that each relation is obtained through decomposition from a relation with attributes *ABCDEFGHI* and that all the known dependencies over relation *ABCDEFGHI* are listed for each question. (The questions are independent of each other, obviously, since the given dependencies over *ABCDEFGHI* are different.) For each (sub)relation: (a) State the strongest nonnal fonn that the relation is in. (b) If it is not in BCNF, decompose it into a collection of BCNF relations.

1. $Rl(A. C,B.D,E)$, $A \rightarrow 13$, $C \rightarrow D$
2. $R2(A,B,F)$, $AC \rightarrow B$, $B \rightarrow F$
3. $R3(A,D)$, $D \rightarrow G$, $G \rightarrow H$
4. $R4(D,C,H,G)$, $A \rightarrow I$, $I \rightarrow A$
5. $R5(A.I,C.B)$

Exercise 19.6 Suppose that we have the following three tuples in a legal instance of a relation schema S with three attributes ABC (listed in order): (1,2,:3), (4,2,3), and (5,3,3).

1. \Which of the following dependencies can you infer does *not* hold over scherna S?

   (a) $A \rightarrow 13$, (b) $Be \rightarrow A$, (c) $13 \rightarrow C$

2. Can you identify allY dependencies that hold over S?

Exercise 19.7 Suppose you are given a relation R with four attributes $ABCD$. For each of the following sets of FDs, assurning those are the only dependencies that hold for R, do the following: (a) Identify the candidate key(s) for R. (b) Identify the best Honnal forBl that R satisfies (1NF, 2NF, 3NF, or BeNF). (c) If R is not in BCNF, decOlnpose it into a set of BCNF relations that preserve the dependencies.

1. $C \rightarrow D$, $C \rightarrow A$. 13 C
2. $B \rightarrow C$. $D \rightarrow A$
3. $ABC \rightarrow D$, $D \rightarrow A$
4. $A \rightarrow B$. $BC \rightarrow D$. $A \rightarrow C$
5. $A13 \rightarrow C$, $AB \rightarrow D$. $C \rightarrow A$, $D \rightarrow 13$

Exercise 19.8 Consider the attribute set $R = ABCDEGH$ and the FD set $F = \{AB \rightarrow C, AC \rightarrow B, AD \rightarrow E, B \rightarrow D, Be \rightarrow A, B \rightarrow G\}$.

1. For each of the following attribute sets, do the following: Cornpute the set of dependencies that hold over the set and write down a rninirnal cover. (ii) Narne the strongest nonnal [onn that is not violated by the relation containing these attributes. (iii) DeCOlnpose it into a collection of BCNF relations if it is ιH)t in BeNF'.

   (a) ABC, (b) $ABCD$, (c) $ABCEG$, (d) $DC:BGII$, (e) $ACEH$

2. Which of the following decOIllpositions of R = $ABCDEG$, with the saIne set of dependencies F, is (a) dependency-preserving? (b) lossless-join?

   (a) $\{AB, BC, ABDE. EG\}$

   (b) $\{ABC, ACDE, ADG\}$

Exercise 19.9 Let R be decOIllposed into $R_1, R_2, \ldots, R_n$. Let F be a set of FDs on R.

1. Define what it rlleans for F to *be pre8erved* in the set of decOlllposed relations.
2. Describe a polynomial-tirne algorithm to test dependency-preservation.
3. Projecting the FDs stated over a set of attributes X onto a subset of attributes Y requires that we consider the closure of the FDs. Give an exarnple where considering the closure is irnportant in testing dependency-preservation, that is, considering just the given FDs gives incorrect results.

Exercise 19.10 Suppose you are given a relation $R(A,B,C,D)$. For each of the following sets of FDs, assuming they are the only dependencies that hold for R, do the following: (a) Identify the candidate key(s) for R. (b) State whether or not the proposed decOlnposition of R into smaller relations is a good decolllposition and briefly explain why or why not.

1. $B \rightarrow C$, $D \rightarrow A$; decornpose into BC and AD.
2. $AB \rightarrow C$, $C \rightarrow A$, $C \rightarrow D$; decompose into ACD and Be.
3. $A \rightarrow BC$, $C \rightarrow AD$; decornpose into ABC and AD.
4. $A \rightarrow B$, B C, $C \rightarrow D$; decornpose into AB and ACD.
5. $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$; decOInpose into AB, AD and CD.

Exercise 19.11 Consider a relation R that has three attributes ABC. It is decornposed into relations R_1 with attributes AB and R_2 with attributes Be.

1. State the definition of a lossless-join decOlnposition with respect to this exarnple. Answer this question concisely by writing a relational algebra equation involving R, R_1, and $H2$.
2. Suppose that $B \rightarrow\rightarrow C$. Is the decorHposition of R into R_1 and $R2$ lossless-join? Reconcile your answer with the observation that neither of the FDs HI nR2 $\rightarrow$ R_I nor R_I n $R_2 \rightarrow R_2$ hold, in light of the simple test offering **a** necessary and sufficient condition for lossless-join decmnposition into two relations in Section 15.6.1.
3. **If** you are given the following justa.nees of R_1 a,nd R_2, what can you say about the instance of R from which these were obtained? Answer this question by listing tuples that are definitely ill R and tuples that are possibly in R.

   Instance of R_1 = {(5,l), (6,l)}
   Instance of R_2 = {(l,8), (1,9)}

   Can you say that attribute B definitely *is* or *is not* a key for R?

Exercise 19.12 Suppose that we have the following four tuples in a relation S with three attributes *ABC:* (1,2,3), (4,2,3), (5,3,3), (5,3,4). Which of the following functional ($\rightarrow$) and rIlultivalued ($\rightarrow\rightarrow$ dependencies can you infer does *not* hold over relation S?

1. $A \rightarrow 13$
2. $A \rightarrow\rightarrow$ B
3. $Be \rightarrow A$
4. $BG \rightarrow\rightarrow A$
5. $8 \rightarrow C$
6. $B \rightarrow\rightarrow$ C

Exercise 19.13 Consider a relation R with five attributes $ABCDE$.

1. For each of the following instances of R, state whether it violates (a) theFD $Be \rightarrow D$ and (b) the MVD $Be \rightarrow\rightarrow D$:
   (a) { } (i.e., mnpty relation)
   (b) {(0,,2,3,4,5), (2,a,3,5,5)}
   (c) {(o,,2,3,4,5), (2,0,,3,5,5), (o,,2,3,4,6)}
   (d) {(a,2,3,4,5), (2,0,,3,4,5), (o,,2,3,6,5)}
   (e) {(0,,2,3,4,5), (2,0,,3,7,5), (a,2,3,4,6)}
   (f) {(o,,2,3,4,5), (2,0,,3,4,5), (0,,2,3,6,5), (o,,2,3,6,6)}
   (g) {(a,2,3,4,5), (0,,2,:3,6,5), (0,,2,3,6,6), (o,,2,3,4,6)}
2. If each instance for R listed above is legal, what can you say about the FD $A \rightarrow B$?

Exercise 19.14 JDs are lllotivated by the fact that sornetilnes a relation that cannot be decoruposed into two sinaller relations in a lossless-join rnanner can be so deC0111pOsed into three or rnore relations. An example is a relation with attributes *supplier*, *part*, and *project*, denoted *SPJ*, with no FDs or MVDs. The JD [xI $\{SP, PJ, JS\}$ holds.

Frorn the JD, the set of relation scheines *SP*, *PJ*, and *JS* is a IORsless-join decornposition of *SPJ*. Construct an instance of *HPJ* to illustrate that no two of these schernes suffice.

Exercise 19.15 Answer the following questions

1. Prove that the algorithrn shown in Figure 19.4 correctly cornputes the attribute closure of the input attribute set X.
2. Describe a linear-tirne (in the size of the set of FI)s, where the size of each FD is the nurnber of attributes involved) algoritlun for finding the attribute closure of a set of attributes with respect to a set of FDs. Prove that your algoritlun correctly COInputes the attribute closure of the input attribute set.

Exercise 19.16 Let us say that an 'Fl) $X \rightarrow Y$ is *simple* if Y is a single attribute.

1. Replace the FD $AB \rightarrow CD$ LJy the srnallest equivalent collection of sirnple FDs.
2. Prove that everyFD $X \rightarrow Y$ in a set of FDs F can be replaced by a set of sirnple F'Ds such that F^+ is equal to the closure of the new set of FDs.

Exercise 19.17 Prove that Arrnstrong's Axioms are sound and complete for FD inference. That is, show **that** repeated application of these axioms on a set F ofF'Ds produces exactly the dependencies in $P+$.

Exercise 19.18 Consider a relation R with attributes $ABCDE$. Let the following FDs be given: $A \rightarrow BC$, $Be \rightarrow E$, and $E \rightarrow DA$. Siluilarly, let S be a relation with attributes $ABCDE$ and let the follo\ving FDs be given: $A \rightarrow BC$, $B \rightarrow E$, and $E \rightarrow DA$. (Only the second dependency differs frolll those that hold over R.) You do not know whether or which other (join) dependencies hold.

1. Is R in BCNF?
2. Is R in 4NF?
3. Is R in 5NF?
4. Is Sin BeNF?
5. Is Sin 4NF?
6. Is Sin 5NF'?

Exercise 19.19 Let R be a relation schelna with a set F of FDs. Prove that the decOlll-position of R into HI and $R2$ is lossless-join if and only if $p+$ contains HI $\cap R_2 \rightarrow R_1$ or $R_1 \cap R2 \rightarrow R_2$.

Exercise 19.20 Consider a scheme R with FDs F that is decOlnposed into schelnes with attributes X and Y. Show that this is dependency-preserving if $F' \subseteq (F_X \cup py)+$.

Exercise 19.21 Prove that the optiInizatioll of the algorithrn for lossless··-join, dependency-preserving decornposition into 3NF relations (Section 19.6.2) is correct.

Exercise 19.22 Prove that the 3NF synthesis algoritlull produces a lossless-join decOlnposition of the relation containing all the original attributes.

Exercise 19.23 Prove that an MVD .X $\rightarrow\rightarrow Y$ over a relation R can be expressed as the join dependency $\bowtie \{XY, X(R - Y)\}$.

Exercise 19.24 Prove that, if R has only one key, it is in BCNF if and only if it is in 3NF.

Exercise 19.25 Prove that, if R is in 3NF and every key is shnple, then R is in HeNF.

Exercise 19.26 Prove these staternents:

1. **If** a relation scherne is in BCNF and at least one of its keys consists of **a** single attrilmte, it is also in 4NF.
2. **If** a relation scherne is in 3NF and each key has a single attribute, it is also in 5NF'.

Exercise 19.27 Give an algorithrn for testing whether a relation scheme is in BCNF. The :.llgorithrn should l)e polynorniaJ in the size of the set of given FDs. (The *size* is the surn over all FI)s of the nurnber of attributes that appear in theFJ).) Is there a polyuOlnial algorithrn for testing whether a relation scheme is in 3NF?

Exercise 19.28 Give an algorithm for testing whether a relation scherne is in **BCNF. The** algorithm should be polynomial in the size of the set of given FI)s. (The 'size' is the SUln over all FI)s of the nurnber of attributes that appear in theFD.) Is there a polynomial algorithrn for testing whether a relation scheme is in 3NF?

Exercise 19.29 1)rove that the algorithm for decomposing a relation scherna with a set of FI)s into a collection of BCNS relation schemas as describerl in Section 19.6.1 is correct (i.e., it produces a collection of BCNF relations, and is lossless-join) and terrninates.

BIBLIOGRAPHIC NOTES

Textbook presentations of dependency theory and its use in database design include [3, 45, 501) 509, 747]. Good survey articles on the topic include [755, 415].

FDs were introduced in [187], along with the concept of 3NF, and aximlls for inferring FDs were presented in [38]. BCNF was introduced in [188]. The concept of a legal relation instance and dependency satisfaction are studied fonnaUy in [328].FDs were generalized to scrnantic data Illodels in [768].

Finding a key is shown to be NP-COlnplete in [497]. Lossless-join decOlnpositions were studied in [28, 502, 627]. Dependency-preserving decorIlpositions were studied in [74]. [81] introduced rninirnal covers. DecOlnposition into 3NF is studied by 1.81, 98] and decOlnposition into BCNF is addressed in [742]. [412] shows that testing whether a relation is in 3NF is NP-cOlnplete. [253] introduced 4NF and discussed decoillposition into 4NF. Fagin introduced other nonnal forIlls in [254] (project-join nonnal fonn) and [255] (doHluin-key nonnal forIll). **In** contrast to the extensive study of vertical decOlnpositions, there has been relatively little formal investigation of horizontal decompositions. [209] investigates horizontal decoillpositiolls.

IvIVDs were discovered independently by Delobel [211] ,Fagin [253], and Zaniolo [789]. AxiOlllS for FDs and MVDs were presented in [73}. [593] shows that there is no axiornatization for JDs, although [662] provides an axioHlatization for a more general class of dependencies. The sufficient conditions for 4NF and 5NF in tenns of FDs that were discussed in Section 19.8 are from [205]. An approach to database design that uses dependency infornation to construct sarnple relation instances is described in [508, 509].