Chapter 13

ACYCLIC DATABASE SCHEMES

In this chapter we introduce a class of database schemes, the *acyclic* database schemes, that possess several desirable properties. We first enumerate the properties, then give three syntactic characterizations of acyclic schemes, give algorithms for two of the characterizations, and prove equivalence of the properties and characterizations.

13.1 PROPERTIES OF DATABASE SCHEMES

In this section we introduce five properties that a database scheme may possess. The properties are mainly "extensional"—they refer to a condition that must hold for all databases on the database scheme. The next section deals with "intensional" properties—ones that involve conditions on the database scheme alone.

13.1.1 Existence of a Full Reducer

We return to the semijoin operator introduced in Chapter 11. Consider a relational expression

$$E = \pi_X(\sigma_C(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_p))$$

where C is some Boolean combination of comparison conditions. Such project-select-join expressions occur frequently as subexpressions when converting calculus-based queries to relational algebra. Suppose we are evaluating the expression on a distributed database system where the relations are spread over multiple sites. It is not unusual in such a system for communication costs between sites to greatly exceed processing costs at a single site. We consider evaluating E while trying to minimize data transmitted between sites, paying no attention to local processing costs.

A naive approach to evaluating E is to ship all the relations to a single site and evaluate the expression at that site. The problem with this approach is that only a small portion of each relation may be needed in the evaluation of E. Tuples and parts of tuples may be excluded from the evaluation by the projection, the selection condition, and the joins. The naive approach can be improved by using the algebraic optimization techniques of Chapter 11 to push parts of the projection and selection down the tree to individual relations. The result is an expression

$$E' = \pi_X(\sigma_{C'}(\pi_{Y_1}(\sigma_{C_1}(r_1)) \bowtie \pi_{Y_2}(\sigma_{C_2}(r_2)) \bowtie \cdots \bowtie \pi_{Y_p}(\sigma_{C_p}(r_p)))).$$

We can compute

$$s_i = \pi_{Y_i}(\sigma_{C_i}(r_i)),$$

for $1 \leq i \leq p$, at individual sites, to be left with the expression

$$E'' = \pi_X(\sigma_{C'}(s_1 \bowtie s_2 \bowtie \cdots \bowtie s_p)$$

to evaluate, where presumably some of the s_i's are smaller then the corresponding r_i's.

Example 13.1 Consider the database $d = \{r_1, r_2, r_3\}$ on the database scheme $\mathbf{R}_a = \{ABC, BCD, CDE\}$ shown in Figure 13.1. ($\mathbf{R}_a$ will be used as a running example throughout this chapter.) Suppose we want to evaluate the expression

$$E = \pi_{AD}(\sigma_{(E<4) \wedge (A \neq D) \wedge (B \neq 8)}(r_1 \bowtie r_2 \bowtie r_3)).$$

We transform E to

$$E' = \pi_{AD}(\sigma_{A \neq D}(\sigma_{B \neq 8}(r_1) \bowtie \sigma_{B \neq 8}(r_2) \bowtie \pi_{CD}(\sigma_{E<4}(r_3)))).$$

We can evaluate

$$s_1 = \sigma_{B \neq 8}(r_1)$$
$$s_2 = \sigma_{B \neq 8}(r_2)$$
$$s_3 = \pi_{CD}(\sigma_{E<4}(r_3))$$

locally to get the database $d' = \{s_1, s_2, s_3\}$ shown in Figure 13.2. The task is now to evaluate

$$E'' = \pi_{AD}(\sigma_{A \neq D}(s_1 \bowtie s_2 \bowtie s_3)).$$

$r_1(A$	B	$C)$
7	4	6
8	4	6
7	5	6
8	8	6
9	8	2
9	4	11
8	5	11

$r_2(B$	C	$D)$
4	6	7
5	6	7
8	6	9
8	11	9
4	11	9
5	11	9
4	12	9

$r_3(C$	D	$E)$
6	7	1
6	7	2
6	7	5
6	9	3
8	7	5
8	9	3
11	9	3
12	7	4

Figure 13.1

$s_1(A$	B	$C)$
7	4	6
8	4	6
7	5	6
9	4	11
8	5	11

$s_2(B$	C	$D)$
4	6	7
5	6	7
4	11	9
5	11	9
4	12	9

$s_3(C$	$D)$
6	7
6	9
8	9
11	9

Figure 13.2

Once the relations have been reduced as far as possible using projection and selections, it may be possible to reduce them further still through semijoins. We are interested in removing all the tuples of the database $d = \{s_1, s_2, \ldots, s_p\}$ that do not participate in the join $s_1 \bowtie s_2 \bowtie \cdots \bowtie s_p$. Recall: The *full reduction* of s_i relative to d, $FR(s_i,d)$, is the set of all tuples of s_i that participate in $\bowtie d$. A *semijoin program SP* is a series of assignments of the form $s_i \leftarrow s_i \ltimes s_j$. *SP* is a *full reducer* relative to the database scheme **R** if $SP(s_i,d) = FR(s_i,d)$ for every database $d(\mathbf{R})$ and relation $s_i \in d$.

If **R** has a full reducer, we can use semijoins to fully reduce $s_1, s_2, \ldots, s_p$ before transmitting them to a common site for joining. Whether or not it pays to apply a particular semijoin program in a distributed system depends on the states of individual relations. In computing $r(R) \bowtie s(S)$ in a distributed system, it could be cheaper to send all of r to s than to send $\pi_{R \cap S}(s)$ to r and then send $r \ltimes s$ back to s. For a given database d, it can happen that one full reducer is beneficial to apply while another is not (see Exercise 13.1).

Example 13.2 The database scheme $\mathbf{R} = \{ABC, BCD, CD\}$ has a full reducer. One full reducer is

$s_2 \leftarrow s_2 \ltimes s_1$;
$s_3 \leftarrow s_3 \ltimes s_2$;
$s_2 \leftarrow s_2 \ltimes s_3$;
$s_1 \leftarrow s_1 \ltimes s_2$.

The result of applying this semijoin program to the relations in Figure 13.2 is shown in Figure 13.3.

$s_1(A$	B	$C)$
7	4	6
8	4	6
7	5	6
9	4	11
8	5	11

$s_2(B$	C	$D)$
4	6	7
5	6	7
4	11	9
5	11	9

$s_3(C$	$D)$
6	7
11	9

Figure 13.3

Example 13.3 Consider the database scheme $\mathbf{R}_c = \{ABC, BCD, CE, DE\}$. ($\mathbf{R}_c$ will also be used for many examples in this chapter.) $\mathbf{R}_c$ has no full-reducers. The database on $\mathbf{R}_c$ shown in Figure 13.4 is not fully reduced, yet no semijoin reduces it further.

$r_1(A$	B	$C)$
1	2	3
7	8	9

$r_2(B$	C	$D)$
2	3	4
8	9	10

$r_3(C$	$E)$
3	5
9	11

$r_4(D$	$E)$
4	11
10	5

Figure 13.4

13.1.2 Equivalence of a Join Dependency to Multivalued Dependencies

Every database scheme **R** corresponds to a unique JD, namely *[**R**]. Every JD implies a set of MVDs. The general implication of one JD by another is given in the next lemma.

Lemma 13.1 If **R** and **S** are database schemes over the same set of attributes, then *[**R**] $\models$ *[**S**] if and only if **S** $\leq$ **R**.*

*Recall that **S** $\geq$ **R** means every relation scheme in **R** is contained in some relation scheme in **S**.

Proof Immediate consequence of Theorem 8.1. $FIX(\mathbf{R}) = SAT(*[\mathbf{R}])$ and $FIX(\mathbf{S}) = SAT(*[\mathbf{S}])$, so $\mathbf{S} \geq \mathbf{R}$ if and only if $FIX(\mathbf{R}) \subseteq FIX(\mathbf{S})$ if and only if $*[\mathbf{R}] \models *[\mathbf{S}]$.

We are interested in the MVDs implied by a JD $*[\mathbf{R}]$. We want to know all pairs of schemes S_1, S_2 such that $*[\mathbf{R}] \models *[S_1, S_2]$. It is sufficient to consider JDs $*[S_1, S_2]$ where S_1 and S_2 are exact unions of schemes in $\mathbf{R}$. In particular we assume there is a function

$$f:\{1, 2, \ldots, p\} \rightarrow \{1,2\}$$

such that

$$S_i = \bigcup_{f(j)=i} R_j, \; i = 1, 2.$$

Let $MVD(\mathbf{R})$ be all the nontrivial MVDs (two-scheme JDs) that can be so defined.

Example 13.4 For $\mathbf{R}_a = \{ABC, BCD, CDE\}$, $MVD(\mathbf{R}_a) = \{*[ABC, BCDE], *[ABCD, CDE]\}$. For $\mathbf{R}_c = \{ABC, BCD, CE, DE\}$, $MVD(R) = \{*[ABC, BCDE], *[ABCD, CDE], *[ABCE5, BCDE]\}$.

Exercise 13.4 shows that any MVD implied by $*[\mathbf{R}]$ is the direct consequence of some MVD in $MVD(\mathbf{R})$. We are interested when $MVD(\mathbf{R}) \models *[\mathbf{R}]$, for a database $\mathbf{R}$. $MVD(\mathbf{R}_a) \models *[\mathbf{R}_a]$, while $MVD(\mathbf{R}_c) \not\models *[\mathbf{R}_c]$ (see Exercise 13.6). Basically, $MVD(\mathbf{R}) \models *[\mathbf{R}]$ means that the lossless decomposition of a relation r onto $\mathbf{R}$ can be captured as a set of two-way decompositions. Also, if $MVD(\mathbf{R}) \models *[\mathbf{R}]$, an efficient test for satisfaction of $*[\mathbf{R}]$ can be devised.

13.1.3 Unique 4NF Decomposition

In this section we formalize the condition that a unique 4NF decomposition follows from a set of MVDs M over a scheme $\mathbf{U}$.

Definition 13.1 Let M be a set of MVDs over a scheme $\mathbf{U}$. A pair of relation schemes (R,S) is a *decomposition* for $\mathbf{U}$ under M if $M \models *[R,S]$. A decomposition (R,S) of $\mathbf{U}$ is *tight* if there is no other decomposition (R',S') with $R' \cap S'$ properly contained in $R \cap S$. That is, (R,S) is tight if the overlap of R and S is minimal.

We are actually interested in decomposing **U** until it is in 4NF. We can view M as applying to a subscheme **U**′ of **U** by considering the MVDs that necessarily apply in $\pi_{\mathbf{U}'}(SAT(M))$. Thus "decomposition under M" and "tight decomposition under M" make sense for subschemes of **U**.

Definition 13.2 Let **R** be a scheme over **U** and let M be a set of MVDs over **U**. **R** is in *tight fourth normal form* (tight 4NF) for M if **R** is in 4NF relative to M and **R** can be obtained by a series of tight decompositions. M *uniquely decomposes* **U** if there is only one database scheme **R** over **U** that is in tight 4NF for M.

Definition 13.3 A database scheme **R** over **U** is a *unique decomposition* if some set M of MVDs uniquely decomposes **U** into **R**.

Example 13.5 Consider database scheme $\mathbf{R}_a = \{ABC, BCD, CDE\}$ from previous examples. $\mathbf{R}_a$ is a unique decomposition of $A\ B\ C\ D\ E$. Let $M = \{BC \twoheadrightarrow A, CD \twoheadrightarrow E\}$ (which is $MVD(\mathbf{R}_a)$). We can either start by decomposing $ABCDE$ into $\{ABC, BCDE\}$ or $\{ABCD, CDE\}$, but at the next step we always reach $\{ABC, BCDE, CDE\}$, which is in 4NF relative to M. $\mathbf{R}_c = \{ABC, BCD, CE, DE\}$ is not a unique decomposition (see Exercise 13.11).

13.1.4 Pairwise Consistency Implies Total Consistency

Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a database scheme and let $d = \{r_1, r_2, \ldots, r_p\}$ be a database over **R**. We have noted in previous chapters that it is computationally hard to test if $r_1, r_2, \ldots, r_p$ join completely. We say that d is *totally consistent* (TC) if $r_1, r_2, \ldots, r_p$ join completely. Database d is *pairwise consistent* (PC) if every pair of relations r_i and r_j join completely. Testing PC is a polynomial computation in the size of a database. TC necessitates PC (Exercise 13.12), but PC is not always sufficient for TC. We are interested in database schemes where every PC database is also TC.

Example 13.6 PC does imply TC for databases on our old friend $\mathbf{R}_a = \{ABC, BCD, CDE\}$. Consider a database $d(\mathbf{R}_a) = \{r_1(ABC), r_2(BCD), r_3(CDE)\}$ that is PC. We show that every tuple in r_2 enters into the join $r_1 \bowtie r_2 \bowtie r_3$. Let t_2 be a tuple in r_2. Since r_1 joins completely with r_2, it contains a tuple t_1 that joins with t_2. Likewise, r_3 contains a tuple t_3 that joins with t_2. The three tuples all join together since $t_1(C) = t_2(C) = t_3(C)$, and C is the only attribute where t_1 and t_3 overlap.

PC is not sufficient for TC for databases on $\mathbf{R}_c = \{ABC, BCD, CE, DE\}$. Figure 13.4 shows a PC database on $\mathbf{R}_c$ that is not TC.

13.1.5 Small Intermediate Joins

Consider the problem of computing $\bowtie d$ for a database $d = \{r_1(R_1), r_2(R_2), \ldots, r_p(R_p)\}$ over scheme R by a series of binary joins. Even if all the relations in d are fully reduced, a poor choice of joins can lead to intermediate results larger than the final result.

Example 13.7 Consider computing $r_1 \bowtie r_2 \bowtie r_3$ for the database on $\mathbf{R}_a = \{ABC, BCD, CDE\}$ shown in Figure 13.5. If we begin by computing $r_1 \bowtie r_3$, we get an intermediate result with 10 tuples, where the complete join has only 6 tuples. If we start with $r_1 \bowtie r_2$, the intermediate result has only 6 tuples.

$r_1(A$	B	$C)$
1	3	5
1	4	5
2	3	5
2	4	6

$r_2(B$	C	$D)$
3	5	7
4	5	8
3	5	9
4	6	8

$r_3(C$	D	$E)$
5	7	10
5	8	10
5	9	11
6	8	11

Figure 13.5

Example 13.8 Consider computing $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$ for the database on $\mathbf{R}_c = \{ABC, BCD, CE, DE\}$ given in Figure 13.6. Any sequence of pairwise joins gives at least one intermediate result with more tuples than the final result (see Exercise 13.14). Note that this database is fully reduced.

$r_1(A$	B	$C)$
1	2	3
1	2	4
1	2	5
1	2	6
1	2	7

$r_2(B$	C	$D)$
2	3	8
2	4	8
2	5	11
2	5	12
2	6	16
2	7	17

$r_3(C$	$E)$
3	9
4	10
5	14
6	15
7	15

$r_4(D$	$E)$
8	9
8	10
11	13
12	14
16	15
17	15

Figure 13.6

We are interested in database schemes where every fully reduced database can be joined through a sequence of pairwise joins where no intermediate result has more tuples than the final result. Moreover, we desire a sequence of joins that works for any database on the scheme and where intermediate results are always "growing." We actually look at a stronger condition, that when a join is taken, the relations involved join completely.

Definition 13.4 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a database scheme. A *join plan* for $\mathbf{R}$ is a rooted binary tree P with leaves labeled by relation schemes in R and every scheme in R labeling at least one leaf of P. Let $d = \{r_1, r_2, \ldots, r_p\}$ be a database on $\mathbf{R}$. The *instantiation of* P *by* d, denoted $P(d)$, is obtained by associating r_i, $1 \leq i \leq p$, with the leaves labeled R_i. After relations are associated with the leaves, associate, recursively, the join of relations at the children with each interior node. The relation $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_p$ is, of course, associated with the root of P.

Example 13.9 Figure 13.7 gives a join plan P for database scheme $\mathbf{R}_a = \{ABC, BCD, CDE\}$. If d is the database in Figure 13.5, Figure 13.8 shows the relations r_a, r_b, and r_c associated with interior nodes, a, b, and c in $P(d)$.

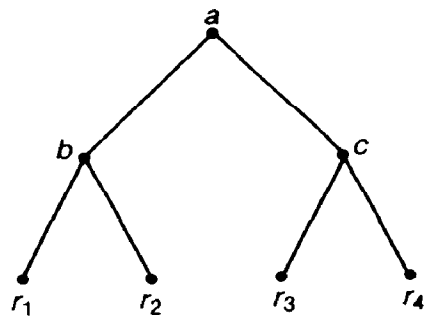

Figure 13.7

$r_a(A$	B	C	D	$E)$
1	3	5	7	10
1	3	5	9	11
1	4	5	8	10
2	3	5	7	10
2	3	5	9	11
2	4	6	8	11

$r_b(A$	B	C	$D)$
1	3	5	7
1	3	5	9
1	4	5	8
2	3	5	7
2	3	5	9
2	4	6	8

$r_c(B$	C	D	$E)$
3	5	7	10
4	5	8	10
3	5	9	11
4	6	8	11

Figure 13.8

Every join plan corresponds to a completely-parenthesized join expression. The join plan in Figure 13.7 corresponds to $(r_1 \bowtie r_2) \bowtie (r_3 \bowtie r_2)$.

Definition 13.5 If P is a join plan for $\mathbf{R}$ and d is a database on $\mathbf{R}$, then $P(d)$ is *monotone* if for every interior node b of P, the relation associated with b is the

complete join of the relations associated with its children. P is *monotone* if $P(d)$ is *monotone* for every PC database d on R.

Example 13.10 Referring back to Example 13.9, $P(d)$ is monotone, and, in fact, P is monotone.

Example 13.11 The join plan P for $\mathbf{R}_c = \{ABC, BCD, CE, DE\}$ given in Figure 13.9 is not monotone. In particular, $P(d)$ is not monotone, where d is the database of Figure 13.6.

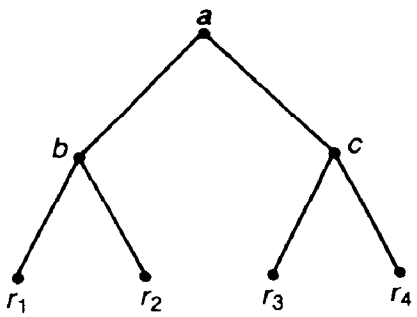

Figure 13.9

Definition 13.6 A database scheme **R** has the *increasing join property* if it has a monotone join plan.

13.2 SYNTACTIC CONDITIONS ON DATABASE SCHEMES

This section introduces three syntactic conditions on database schemes: acyclicity, existence of a join tree, and the running intersection property. In the next section we introduce algorithms for testing two of these conditions. We also demonstrate there the equivalence of the syntactic conditions of this section and the more extensional properties of the last section.

13.2.1 Acyclic Hypergraphs

A hypergraph is similar to an ordinary undirected graph, except that edges are arbitrary nonempty sets of nodes, rather than just doubletons.

Definition 13.7 A *hypergraph* H is a pair $(\mathfrak{N},\mathcal{E})$ where $\mathfrak{N}$ is a set of items, called *nodes*, and $\mathcal{E}$ consists of nonempty subsets of $\mathfrak{N}$, called *hyperedges*. If it is clear we are dealing with hypergraphs, we may use "edges" for "hyperedges." H is *reduced* if no edge in $\mathcal{E}$ properly contains another edge and every node is in some edge. The *reduction* of H, written $RED(H)$, is H with any contained edges and non-edge nodes removed.

A database scheme is naturally viewed as a hypergraph. If $\mathbf{R}$ is a database scheme over $\mathbf{U}$, then $\mathbf{R}$ may be viewed as the hypergraph $(\mathbf{U},\mathbf{R})$. That is, the attributes in $\mathbf{R}$ are the nodes in the hypergraph and the relation schemes of $\mathbf{R}$ are the hyperedges. We shall simply use $\mathbf{R}$ in place of $(\mathbf{U},\mathbf{R})$ when dealing with the hypergraph that $\mathbf{R}$ represents. Saying that $\mathbf{R}$ is reduced is saying that $\mathbf{R}$ is reduced as a hypergraph: no relation scheme in $\mathbf{R}$ properly contains another.

Example 13.12 In drawing hypergraphs, nodes are represented by their labels and hyperedges are represented by closed curves around the nodes. The hypergraph for $\mathbf{R}_a = \{ABC, BCD, CDE\}$ is given in Figure 13.10. The hypergraph for $\mathbf{R}_c = \{ABC, BCD, CE, DE\}$ is given in Figure 13.11.

Definition 13.8 Let $H = (\mathfrak{N},\mathcal{E})$ be a hypergraph, with A and B nodes in $\mathfrak{N}$. A *path* from A to B in H is a sequence of edges $E_1, E_2, \ldots, E_k$, $k \leq 1$, such that $A \in E_1$, $B \in E_k$ and $E_i \cap E_{i+1} \neq \emptyset$ for $1 \leq i < k$. We also say that E_1, $E_2, \ldots, E_k$ is a path from E_1 to E_k.

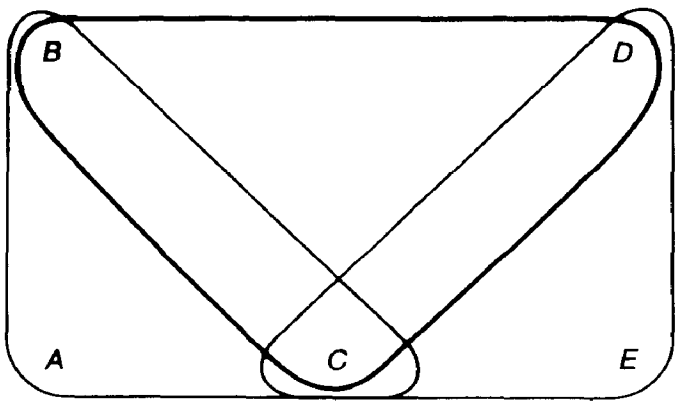

Figure 13.10

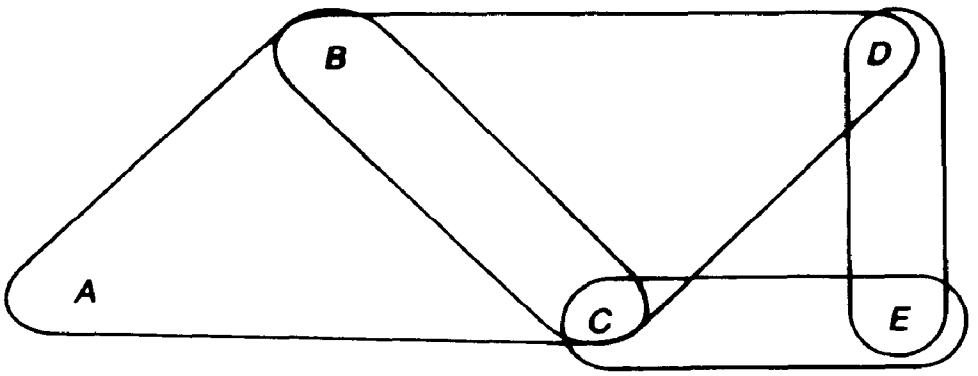

Figure 13.11

Definition 13.9 In a hypergraph $H = (\mathfrak{N},\mathcal{E})$, two nodes or edges are *connected* if there is a path between them. A set of edges is *connected* if every pair of edges is connected. A *connected component* of H is a maximal connected set of edges.

Example 13.13 Let H be the hypergraph shown in Figure 13.12. ABC, BCD, DE is a path from A to E and from ABC to DE, so A and E are connected, as are ABC and DE. The connected components of H are $\{ABC, BCD, DE\}$ and $\{IJ, JKL, IKL\}$.

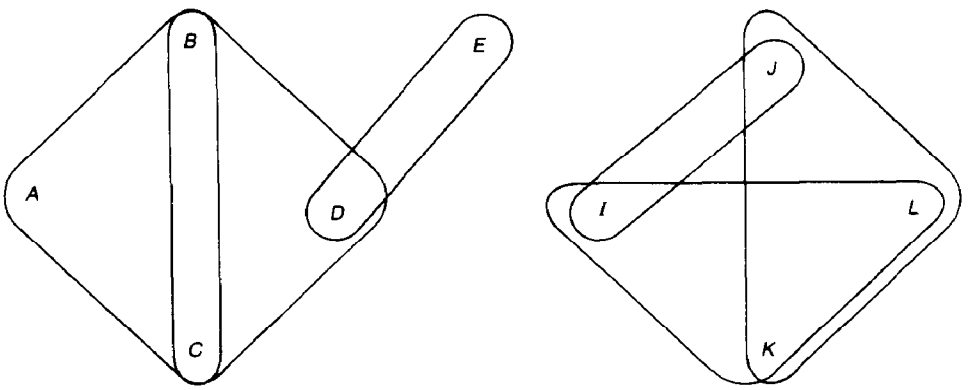

Figure 13.12

We shall be concerned mainly with hypergraphs that consist of a single connected component. Most of what we do generalizes to hypergraphs with multiple components.

Definition 13.10 Let $H = (\mathfrak{N}, \mathcal{E})$ and $H' = (\mathfrak{N}', \mathcal{E}')$ be hypergraphs. H' is a *subhypergraph* of H if $\mathfrak{N}' \subseteq \mathfrak{N}$ and $\mathcal{E}' \subseteq \mathcal{E}$.

Definition 13.11 Let $H = (\mathfrak{N}, \mathcal{E})$ be a hypergraph and let $\mathfrak{M} \subseteq \mathfrak{N}$. The *$\mathfrak{M}$-induced hypergraph for* H, denoted $H_{\mathfrak{M}}$, is the hypergraph $RED((\mathfrak{M}, \mathcal{E}_{\mathfrak{M}}))$ where

$$\mathcal{E}_{\mathfrak{M}} = \{E \cap \mathfrak{M} | E \in \mathcal{E}\}.$$

$H_{\mathfrak{M}}$ is not necessarily a subhypergraph of H, since $\mathcal{E}_{\mathfrak{M}}$ may contain edges not in $\mathcal{E}$.

Example 13.14 Let H be the hypergraph $(ABCDEIJK, \{ABC, BD, CDE, DEI, IJK\})$ shown in Figure 13.13. $H' = (ABCDE, \{ABC, BD, CDE\})$ is a subhypergraph of H, as well as being the $ABCDE$-induced hypergraph for H. $H_{ABCD} = (ABCD, \{ABC, BD, CD\})$, as shown in Figure 13.14. H_{ABCD} is not a subhypergraph of H, since CD is not an edge of H.

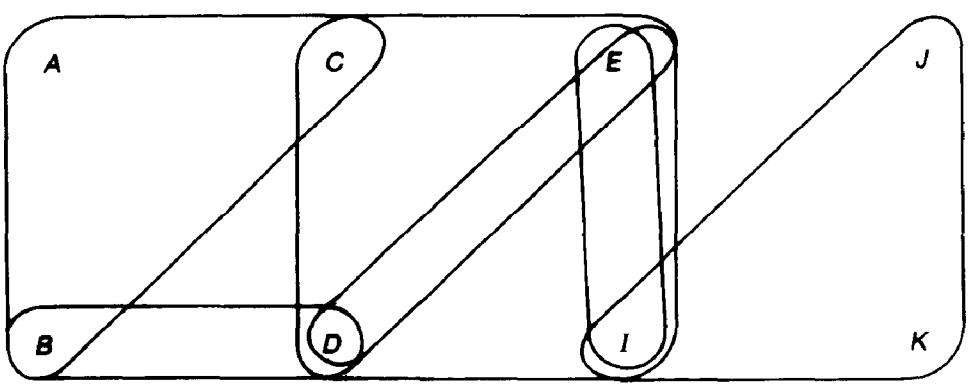

Figure 13.13

We now wish to generalize the notion of "strongly connected" from ordinary graphs to hypergraphs. Recall that a strongly connected graph is one with no articulation points.

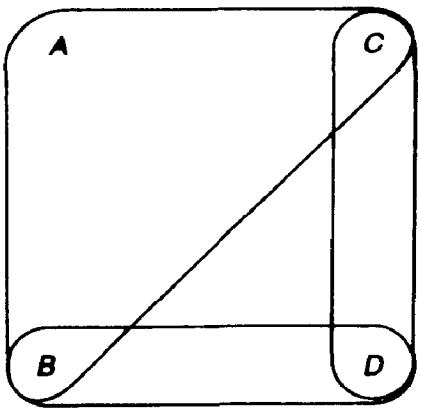

Figure 13.14

Definition 13.12 Let $H = (\mathfrak{N}, \mathcal{E})$ be a hypergraph. A set $F \subseteq \mathfrak{N}$ is an *articulation set* for H if $F = E_1 \cap E_2$ for some pair of edges $E_1, E_2 \in \mathcal{E}$, and $H_{\mathfrak{N}'}$ has more connected components than H, where $\mathfrak{N}' = \mathfrak{N} - F$. That is, removing the nodes in F from H disconnects some pair of nodes that were previously connected in H.

Definition 13.13 Let $H = (\mathfrak{N}, \mathcal{E})$ be a hypergraph. A *block* of H is an $\mathfrak{M}$-induced hypergraph of H with no articulation set, for some $\mathfrak{M} \subseteq \mathfrak{N}$. A block is *trivial* if it has only one edge. A reduced hypergraph is *acyclic* if it has no blocks; otherwise it is *cyclic*. An arbitrary hypergraph is cyclic or acyclic precisely when its reduction is.

Example 13.15 Let H be the hypergraph of Example 13.14. DE is an articulation set of H, since $DE = CDE \cap DEI$, and H_{ABCIJK} has two components where H had one. H_{ABCD}, shown in Figure 13.14, is a block of H, since it contains no articulation set. Since H is reduced, we conclude it is cyclic.

Example 13.16 Consider the database scheme $\mathbf{R}_a = \{ABC, BCD, CDE\}$ as a hypergraph. $\mathbf{R}_a$ is acyclic. For example, consider $(\mathbf{R}_a)_{ABDE}$, shown in Figure 13.15. It is not a block because it has B and D as articulation sets.

We now give a slightly different definition of acyclicity that only considers induced hypergraphs that are subhypergraphs.

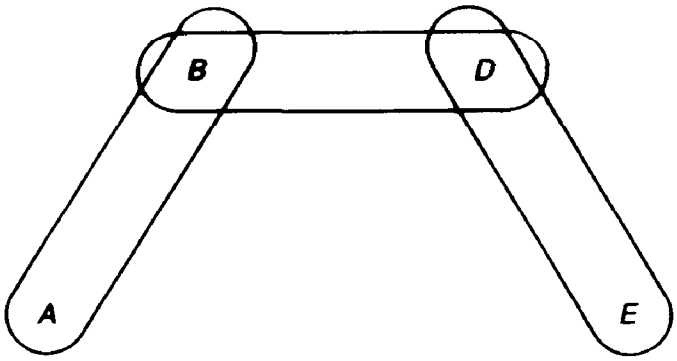

Figure 13.15

Definition 13.14 Let $H = (\mathfrak{N}, \mathcal{E})$ be a hypergraph and let $H' = (\mathfrak{N}', \mathcal{E}')$ be a subhypergraph of H. H' is *closed* relative to H if $H' = H_{\mathfrak{M}}$ for some $\mathfrak{M} \subseteq \mathfrak{N}$. Clearly if such an $\mathfrak{M}$ exists, it must be $\mathfrak{N}'$. Equivalently, H' is a closed subhypergraph of H if for any edge $E \in \mathcal{E}$ there is an edge $E' \in \mathcal{E}'$ such that $E' \supseteq \mathfrak{N}' \cap E$.

Definition 13.15 A reduced hypergraph H is *closed-acyclic* if every closed, connected subhypergraph of H with two or more edges has an articulation set; otherwise H is *closed-cyclic*. An arbitrary hypergraph is closed-acyclic and closed-cyclic exactly as its reduction is.

Example 13.17 Let H be the hypergraph of Example 13.14. $H' = (ABCD, \{ABC, BD\})$ is not closed relative to H. Consider the edge CDE of H. $CDE \cap ABCD = CD$, and CD is not contained in any edge of H'. $H'' = (ABCDE, \{ABC, BD, CDE\})$ is closed relative to H, since $H'' = H_{ABCDE}$. H'' is pictured in Figure 13.16. H'' has no articulation set, so H is closed-cyclic.

Acyclic and closed-acyclic are equivalent conditions (see Exercise 13.22). A database scheme $\mathbf{R}$ is *acyclic* if $\mathbf{R}$ considered as a hypergraph is acyclic.

13.2.2 Join Trees

Definition 13.16 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a database scheme over $\mathbf{U}$. The *complete intersection graph* for $\mathbf{R}$, denoted $I_{\mathbf{R}}$, is the complete undirected graph on nodes $R_1, R_2, \ldots, R_p$ and with edge labels chosen from the subsets

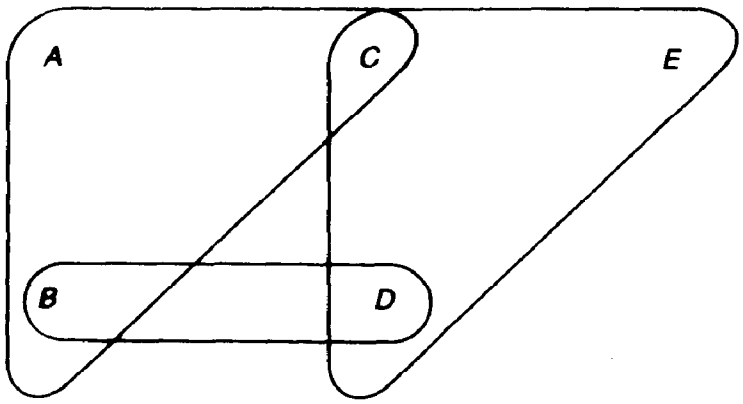

Figure 13.16

of **U**. For an edge $e = (R_i, R_j)$, the label of e, denoted $L(e)$, is $R_i \cap R_j$. An *intersection graph* for **R** is any subgraph of $I_{\mathbf{R}}$ formed by removing only edges. In drawing an intersection graph, we generally omit any edge e where $L(e) = \emptyset$.

Definition 13.17 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a database scheme over **U**. Let G be an intersection graph for R and let $A \in \mathbf{U}$. A path $e_1, e_2, \ldots, e_k$ from node R_i to node R_j in G is an A-*path* if $A \in L(e_i)$ for all $1 \leq i \leq k$. If $e_1, e_2, \ldots, e_k$ is an A-path, then it follows $A \in R_i$ and $A \in R_j$. In fact, A must be in every node R along the A-path.

Example 13.18 Consider the database scheme $\mathbf{R} = \{ABC, BD, CDE, DEI, IJK\}$ over $ABCDEIJK$, which corresponds to the hypergraph of Example 13.14. $I_{\mathbf{R}}$ is shown in Figure 13.17 (omitting edges with empty labels). Figure 13.18 gives an intersection graph G for **R**. G has a D-path from BD to CDE. There is no B-path in G from ABC to BD, although $I_{\mathbf{R}}$ has such a path.

Definition 13.18 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a database scheme over **U**. An intersection graph G for **R** is a *join graph* if for every pair of nodes R_i, R_j in G, if $A \in R_i \cap R_j$ then there is an A-path from R_i to R_j. A *join tree* is a join graph that is a tree.

Example 13.19 Let **R** be the database scheme from Example 13.18. $I_{\mathbf{R}}$ is a join graph for **R**. (The complete intersection graph is always a join graph.)

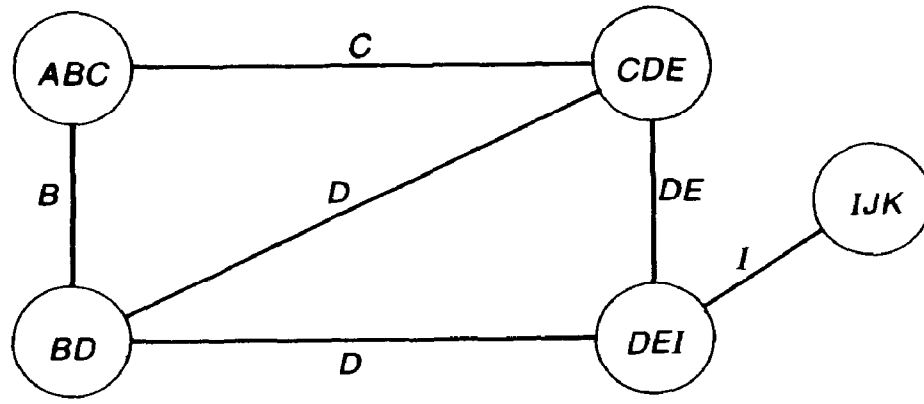

Figure 13.17

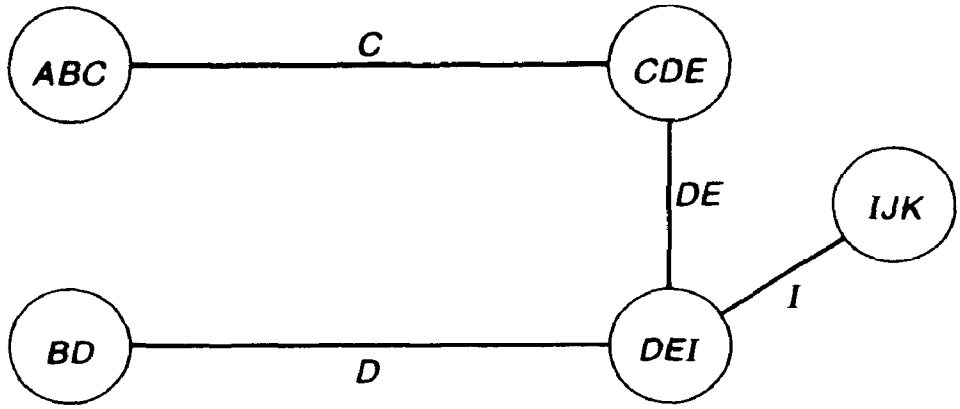

Figure 13.18

The intersection graph in Figure 13.18 is not a join graph for **R**, since there is no B-path from ABC to BD. **R** has no join trees. Any join graph G for R must have the edge (ABC,BD) to give a B-path from ABC to BD, as well as the edge (ABC,CDE) to give a C-path from ABC to CDE. Nodes BD and CDE must be connected by a D-path. The D-path cannot go through ABC, so G must contain a cycle.

We are interested in database schemes where join trees exist. We shall see later that a join tree can be used to construct monotone join plans.

Example 13.20 The database scheme $\mathbf{R}_a = \{ABC, BCD, CDE\}$ does have a join tree, as shown in Figure 13.19.

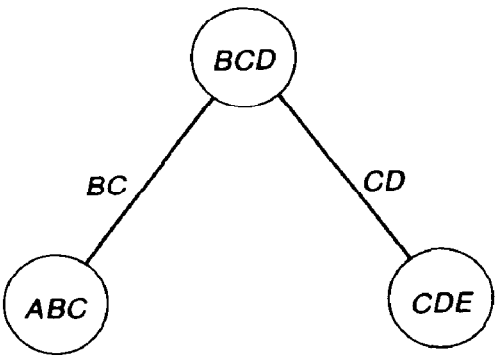

Figure 13.19

13.2.3 The Running Intersection Property

Definition 13.19 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a database scheme. $\mathbf{R}$ has the *running intersection property* if there is a permutation $S_1, S_2, \ldots, S_p$ of $R_1, R_2, \ldots, R_p$ such that for every $1 < i \leq p$, there exists a $j < i$ such that

$$(S_1 S_2 \cdots S_{i-1}) \cap S_i \subseteq S_j.$$

That is, the intersection of S_i with the union of all the previous schemes is contained entirely within one of those schemes.

Example 13.21 $\mathbf{R} = \{ABC, CDE, BCEI\}$ has the running intersection property, as witnessed by the ordering $BCEI, ABC, CDE$ of its relation schemes.

13.3 EQUIVALENCE OF CONDITIONS

As was remarked at the beginning of the chapter, and as the running examples $\mathbf{R}_a$ and $\mathbf{R}_c$ indicate, all the properties and conditions in Sections 13.1 and 13.2 describe the same class of database schemes. Before proving the equivalences, we look at algorithms to decide whether a database scheme $\mathbf{R}$ is acyclic and whether $\mathbf{R}$ has a join tree.

13.3.1 Graham Reduction

The following algorithm on hypergraphs was introduced by Graham, although Yu and Ozsoyoglu independently gave an essentially equivalent algorithm that runs on a different data structure. The *Graham reduction algorithm* consists of repeated application of two reduction rules to hypergraphs until neither can be applied further. Let $H = (\mathfrak{N}, \mathcal{E})$ be a hypergraph. The two reduction rules are

rE. (edge removal) If E and F are edges in $\mathcal{E}$ such that E is properly contained in F, remove E from $\mathcal{E}$.
rN. (node removal) If A is a node in $\mathfrak{N}$, and A is contained in at most one edge in $\mathcal{E}$, remove A from $\mathfrak{N}$ and also from all edges in $\mathcal{E}$ in which it appears.

Example 13.22 Figure 13.20 shows the stages in applying the Graham reduction algorithm to the hypergraph for $\mathbf{R}_a = \{ABC, BCD, CDE\}$. The labeled arrows represent applications of the corresponding reduction rule.

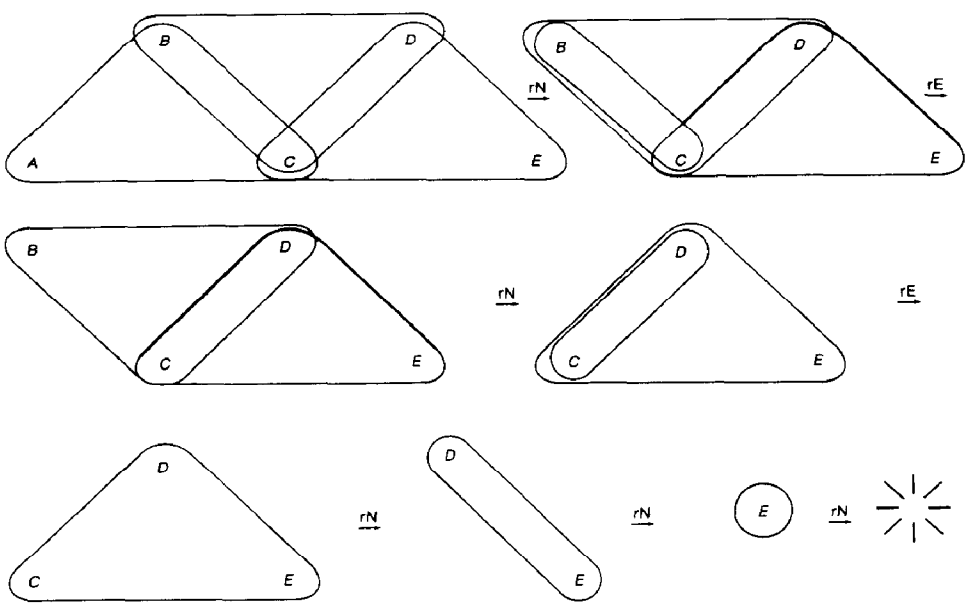

Figure 13.20

Example 13.23 Figure 13.21 shows the stages in applying Graham reduction to $\mathbf{R}_c = \{ABC, BCD, CE, DE\}$.

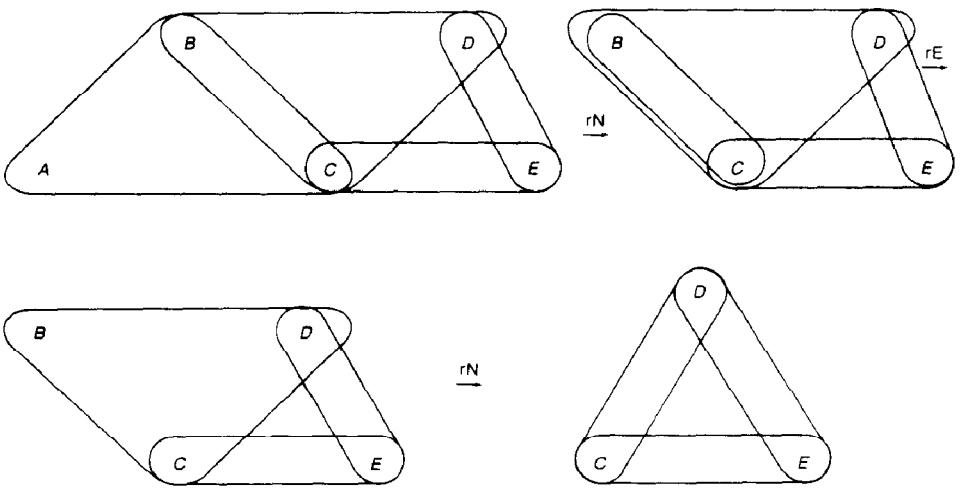

Figure 13.21

We say the Graham reduction *succeeds* on hypergraph H if the result of applying the Graham reduction algorithm to H is the empty hypergraph, as for $\mathbf{R}_a$ above.

13.3.2 Finding Join Trees

In this section we assume the reader is familiar with algorithms for finding minimum-weight spanning trees of undirected graphs with weighted edges. We shall actually be interested in finding *maximum-weight* spanning trees. Since all spanning trees for a graph have the same number of edges, an algorithm for finding a minimum-weight spanning tree can be converted to an algorithm for maximum-weight spanning trees by negating edge weights.

For the following definitions we assume a database scheme $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ over $\mathbf{U}$ and an intersection graph G for $\mathbf{R}$.

Definition 13.20 For attribute $A \in \mathbf{U}$, the *class of* A, denoted $CLASS(A)$, is $\{R_i | A \in R_i \text{ and } R_i \in \mathbf{R}\}$. The *weight of* A, denoted $WT(A)$, is $|CLASS(A)| - 1$. The *weight of* $\mathbf{R}$, $WT(\mathbf{R})$, is

$$\sum_{A \in \mathbf{U}} WT(A).$$

Definition 13.21 The *weight of* A *in* G, denoted $WT(A,G)$ is the number of edges in G that contain A in their labels. The *weight of* G, denoted $WT(G)$, is

$$\sum_{A \in \mathbf{U}} WT(A,G).$$

Definition 13.22 For an edge e in G, the *weight of* e, denoted $WT(e)$, is $|L(e)|$. Observe that $WT(G)$ could also be computed as

$$\sum_{e \in G} WT(e)$$

Example 13.24 Let G be the join graph in Figure 13.22 for the database scheme $\mathbf{R} = \{ABC, BD, CDE, DEI, IJK\}$. For $\mathbf{R}$,

$$\begin{array}{llll} WT(A) = 0 & & WT(E) = 1 \\ WT(B) = 1 & & WT(I) = 1 \\ WT(C) = 1 & & WT(J) = 0 \\ WT(D) = 2 & & WT(K) = 0 \end{array}$$

and so $WT(\mathbf{R}) = 6$. For G,

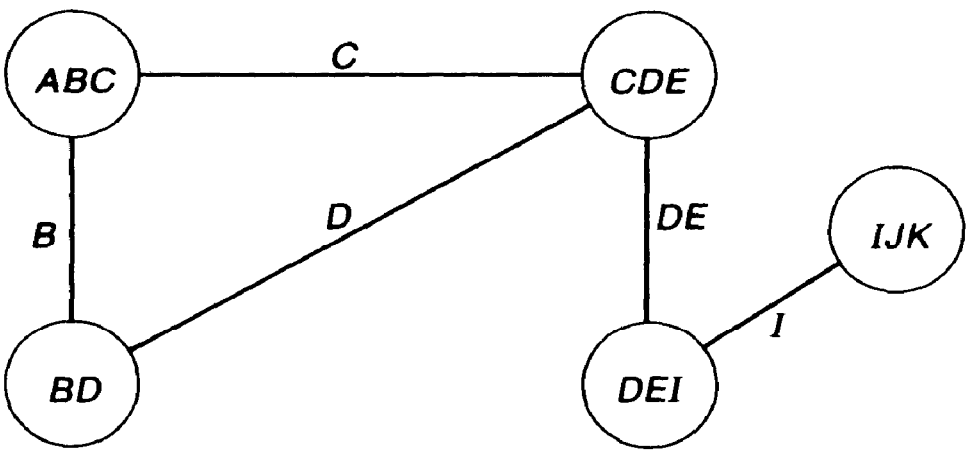

Figure 13.22

$$\begin{aligned} WT(A,G) &= 0 & WT(E,G) &= 1 \\ WT(B,G) &= 1 & WT(I,G) &= 1 \\ WT(C,G) &= 1 & WT(J,G) &= 0 \\ WT(D,G) &= 2 & WT(K,G) &= 0 \end{aligned}$$

and so $WT(G) = 6$.

Theorem 13.1 If a database scheme $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ has a join tree G, then any maximum-weight (edge weight) spanning tree for $I_{\mathbf{R}}$ is a join tree. Furthermore, G is a maximum-weight spanning tree for $I_{\mathbf{R}}$ and $WT(G) = WT(\mathbf{R})$.

Proof First, we show that in G, $WT(A)$ must equal $WT(A,G)$ for any attribute A. There are $WT(A) + 1$ nodes in G that contain A. It requires at least $WT(A)$ edges to construct A-paths between every pair. Hence, $WT(A,G) \geq WT(A)$. Any edge e with $A \in L(e)$ must connect elements of $CLASS(A)$. If G contained more than $WT(A)$ edges with A in their label, those edges would form a cycle among some set of the nodes in $CLASS(A)$. Hence $WT(A,G) \leq WT(A)$, so $WT(A,G) = WT(A)$. It follows that $WT(G) = WT(\mathbf{R})$.

G is a spanning tree for $I_{\mathbf{R}}$. Suppose there is another spanning tree G' for $I_{\mathbf{R}}$ with weight greater than G. There must be an attribute A with $WT(A,G) < WT(A,G')$. By the remarks in the last paragraph, G' must contain a cycle among some nodes in $CLASS(A)$, contradicting the choice of G'. G must be a maximum-weight spanning tree.

Finally, let G' be any maximum-weight spanning tree of $I_{\mathbf{R}}$. By previous arguments, for any attribute A, $WT(A) = WT(A,G) = WT(A,G')$. Since G' is a tree, and there are $WT(A)$ edges with A in their label in G', any two members of $CLASS(A)$ must be connected by an A-path in G'. Hence G' is a join tree.

Theorem 13.1 gives a reasonably efficient test for the existence of join trees for a database scheme R. Find $I_{\mathbf{R}}$ (only edges with non-empty labels are necessary) and then find a maximum-weight spanning tree G for $I_{\mathbf{R}}$. If G is a join tree, then, obviously, $\mathbf{R}$ has a join tree. If G is not a join tree, then $\mathbf{R}$ has no such tree.

Example 13.25 Figure 13.18 shows a maximum-weight spanning tree G for $I_{\mathbf{R}}$, where $\mathbf{R} = \{ABC, BD, CDE, DEI, IJK\}$. G is not a join tree, so $\mathbf{R}$ has no join tree, as was noted before.

Example 13.26 Figure 13.19 shows a maximum-weight spanning tree G for $I_{\mathbf{R}_a}$, where $\mathbf{R}_a = \{ABC, BCD, CDE\}$. As noted before, G is a join tree for $\mathbf{R}_a$.

13.3.3 The Equivalence Theorem for Acyclic Database Schemes

Theorem 13.2 Let $\mathbf{R}$ be a connected database scheme. The following conditions are equivalent:

1. $\mathbf{R}$ is acyclic
2. Graham reduction succeeds on $\mathbf{R}$.
3. $\mathbf{R}$ has a join tree.
4. $\mathbf{R}$ has a full reducer.
5. PC implies TC for $\mathbf{R}$.
6. $\mathbf{R}$ has the running intersection property.
7. $\mathbf{R}$ has the increasing join property.
8. $RED(\mathbf{R})$ is a unique 4NF decomposition.
9. The maximum weight spanning tree for $I_{\mathbf{R}}$ is a join tree.
10. $MVD(\mathbf{R}) \models *[\mathbf{R}]$.

Proof The proof will proceed via a series of lemmas. The equivalence of 3 and 9 was established in Theorem 13.1. The method for the rest of the equivalence is $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 5 \Rightarrow 1, 3 \Rightarrow 6 \Rightarrow 7 \Rightarrow 5, 8 \Leftrightarrow 1, 3 \Rightarrow 10$. The implication of any other condition by 10 is left as Exercise 13.36. The lemmas for these implications are shown in Figure 13.23.

Lemma 13.2 If $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ is an acyclic database scheme, then Graham reduction succeeds on $\mathbf{R}$.

The proof of Lemma 13.2 proceeds through four propositions. The first two show that the Graham reduction algorithm neither creates nor destroys blocks. The second two show that one of the two removal rules is always applicable to an acyclic hypergraph.

Proposition 13.1 The Graham reduction algorithm preserves blocks.

Proof Let $H = (\mathfrak{N}, \mathcal{E})$ be a hypergraph such that $H_{\mathfrak{M}}$ is a block for some $\mathfrak{M} \subseteq \mathfrak{N}$. Let H' be obtained from H by one application of rE (edge removal). $H_{\mathfrak{M}}$ must be the same as $H'_{\mathfrak{M}}$ because reduction is applied in forming an induced hypergraph. If $E \subseteq F$ is the edge removed, then $E \cap \mathfrak{M} \subseteq F \cap \mathfrak{M}$, so E makes no contribution to $H_{\mathfrak{M}}$.

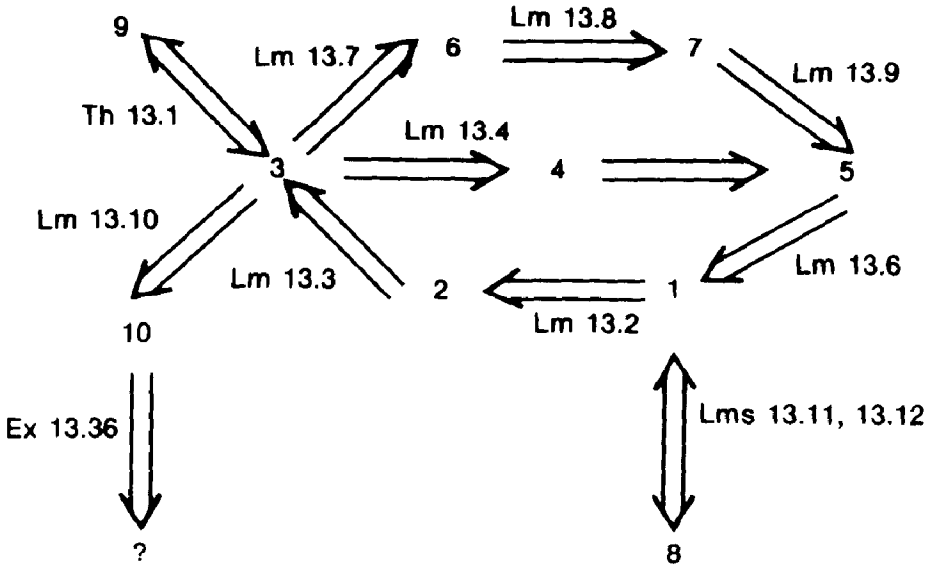

Figure 13.23

Suppose now that rule rN (node removal) was used on node A to obtain H'. If $A \notin \mathfrak{M}$, and $H_{\mathfrak{M}}$ is a block, then so is $H'_{\mathfrak{M}}$. If $A \in \mathfrak{M}$, and $H_{\mathfrak{M}}$ is a block, we must show $H'_{\mathfrak{M}-A}$ is a block. If $F = E_1 \cap E_2$ is an articulation set of $H'_{\mathfrak{M}-A}$, then it must also be an articulation set of $H_{\mathfrak{M}}$. E_1 or E_2 could be augmented by A in $H_{\mathfrak{M}}$, but not both, since A appears in at most one edge of H. It follows F is the intersection of edges in $H_{\mathfrak{M}}$. If removing F disconnects $H'_{\mathfrak{M}-A}$, it will also disconnect H_M, since A cannot contribute to connectivity. We conclude that if H does not have an articulation set, neither does $H_{\mathfrak{M}}$.

Application of either rE or rN preserves blocks, so Graham reduction preserves blocks.

Proposition 13.2 Graham reduction does not introduce blocks.

Proof Let $H = (\mathfrak{N}, \mathcal{E})$ be a hypergraph. As noted in the last proof, if H' is obtained from H by rE, then $H_{\mathfrak{M}} = H'_{\mathfrak{M}}$ for any $\mathfrak{M} \subseteq \mathfrak{N}$. Hence, rE cannot introduce blocks.

Suppose H' is obtained from H by removing node A according to rN. Suppose $H_{\mathfrak{M}}$ has an articulation set while $H'_{\mathfrak{M}-A}$ is a block. $H_{\mathfrak{M}-A} = H'_{\mathfrak{M}-A}$, so H had a block to begin with. Since neither rE nor rN introduce blocks, Graham reduction does not introduce blocks.

If F is an articulation set of hypergraph $H = (\mathfrak{N}, \mathcal{E})$, we say F *splits* H into subhypergraphs $H_1, H_2, \ldots, H_k$ if each H_i is one of the connected components in $H_{\mathfrak{N}-F}$ with its partial edges augmented back to full edges by the addition of nodes from F. Note that $H_1, H_2, \ldots, H_k$ share no edges.

Example 13.27 BC is an articulation set for $H = (ABCDEIJ, \{ABC, BCD, BEI, CEJ\})$. BC splits H into

$H_1 = (ABC, \{ABC\})$,
$H_2 = (BCD, \{BCD\})$, and
$H_3 = (BCEIJ, \{BEI, CEJ\})$.

Proposition 13.3 Let $H = (\mathfrak{N}, \mathcal{E})$ be an acyclic hypergraph where $|\mathcal{E}| \geq 2$ and such that H is connected. H has an articulation set F that splits H into subhypergraphs $H_1, H_2, \ldots, H_k$ where each H_i contains an edge E_i with $F \subseteq E_i$. It follows that each H_i is a closed subhypergraph of H.

Proof Let H be an acyclic hypergraph with fewest nodes that violates the lemma. Let $F = E_1 \cap E_2$ be any articulation set of H that does not properly contain another articulation set. Let F split H into subhypergraphs $H_1, H_2, \ldots, H_k$. Suppose, without loss of generality, that $H_1 = (\mathfrak{N}_1, \mathcal{E}_1)$ contains no edge containing F. Form a subhypergraph H' of H where $H' = (\mathfrak{N}_1 \cup E_1, \mathcal{E} \cup \{E_1\})$. Any edge outside H' that intersects $\mathfrak{N}_1 \cup E_1$ must do so within E_1, so H' is closed with respect to H. It follows that H' is the $(\mathfrak{N}_1 \cup E_1)$-induced hypergraph for H.

Since H is acyclic, and H' is node-induced, H' must be acyclic (see Exercise 13.27). H' is smaller than H, so it has an articulation set $F' = E_3 \cap E_4$ that splits H' into subhypergraphs $H_1', H_2', \ldots, H_m'$ such that every H_i' contains an edge containing F'. We claim that F' is an articulation set for all of H. Let H_1' be the subhypergraph of H' containing E_1. If any edge E of H outside of H' touches $H_2', H_3', \ldots, H_m'$ outside of E_1, F could not have split off H_1 in the first place.

We further claim that F' splits H into $H_1'', H_2', H_3', \ldots, H_m'$, where H_1'' is H_1' plus all the nodes and edges from $H_2, H_3, \ldots, H_k$. That is, H_1'' is H_1' plus all of H outside of H_1. Certainly, F' splits $H_2', H_3', \ldots, H_m'$ from H. Can F' split the rest of H into more than one subhypergraph? All of $H_2, H_3, \ldots, H_k$ touch H_1', since they all touch E_1 in H_1'. Consider F' relative to F. If $F' \supseteq F$, then both E_3 and E_4 contain F, and H_1 would have had an edge containing F.

We must have that $F' \cap E_1 \subsetneq F$. $F' \cap E_1$ disconnects part of H outside of H_1, and $F' \cap E_1 = E_3 \cap E_1$ or $E_4 \cap E_1$, contradicting the minimality of F.

We have shown that if F does not meet the requirements of the lemma, then F' does, because each of H_1'', H_2', ..., H_m' contain an edge containing F'.

Definition 13.23 An edge E in hypergraph H is a *knob* if E contains at least one node contained in no other edge of H. Such a node is called a *solitary node*.

Example 13.28 In the hypergraph $H_1 = (ABCDE, \{ABC, BCD, CDE\})$, both ABC and CDE are knobs. A and E are solitary nodes. The hypergraph $H_2 = (ABCDE, \{ABC, BCD, CDE, ADE\})$ has no knobs.

Proposition 13.4 Any reduced, acyclic hypergraph H with two or more edges has at least two knobs.

Proof The proposition is clearly true for any reduced, acyclic hypergraph $H = (\mathfrak{N}, \mathcal{E})$ where $|\mathcal{E}| = 2$. Assume the proposition holds when $|\mathcal{E}| = k - 1$ and consider the case where $|\mathcal{E}| = k$. Let F be an articulation set of H as guaranteed in Proposition 13.3. Let F split H into $H_1, H_2, \ldots, H_k$. Each H_i is closed with respect to H, hence node-induced, hence acyclic. Since H_i contains only edges from H, it is reduced.

Consider H_1. If H_1 has more than one edge, then, by induction, it has two knobs. Since some edge E_1 in H_1 contains F, at most one knob of H_1 can have all its solitary nodes contained in F. The other knob cannot intersect $H_2, H_3, \ldots, H_k$ outside of F, or else F would not have split off H_1. Thus, the other knob is a knob for H. If H_1 is a single edge, that edge is a knob for H.

Since the same argument holds for H_2, H has two knobs.

Proof of Lemma 13.2 By Proposition 13.2, Graham reduction preserves acyclicity. At any point in Graham reduction of an acyclic hypergraph H, if H is not reduced, rE can be applied. If the intermediate result is reduced, Proposition 13.4 holds, or we are down to a single edge, so rN can be used to remove a solitary node. Since in Graham reduction, an application of a removal rule reduces the number of nodes or edges, the algorithm must eventually succeed in reducing H to the empty hypergraph.

Graham reduction cannot succeed on a cyclic hypergraph H. H must have at least three edges. If Graham reduction succeeded on H, there must have been an intermediate result with just two edges, which must therefore have been acyclic. Such an intermediate result contradicts Proposition 13.1.

Lemma 13.3 If Graham reduction succeeds on the hypergraph for a connected database scheme **R**, then **R** has a join tree.

Proof Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$. Running Graham reduction on **R** will never disconnect **R**. We build a join tree G for **R** as follows. Let $REM_j(R_i)$ be what remains of R_i before the j^{th} step of the reduction. If the j^{th} step applies rE to remove $REM_j(R_i)$ because it is contained in $REM_j(R_k)$, add edge (R_i, R_k) to G with label $R_i \cap R_k$.

The resulting graph G is clearly an intersection graph. G can have no cycles. Each node R_i in G is connected to at most one node R_k in G such that the remainder of R_k was removed after the remainder of R_i. Any cycle must contain an R_i connected to two nodes whose remainders were removed after the remainder of R_i in Graham reduction. G is a tree by the connectivity remark above.

Is G a join graph? Suppose not. Renumber the schemes in **R** so that there is an attribute $A \in R_1 \cap R_2$ but there is no A-path from R_1 to R_2 in G. Assume further that R_1 and R_2 were chosen so as to minimize the distance between them in the tree G. Finally, assume the remainder of R_1 was removed before the remainder of R_2. At some step j, we must have $REM_j(R_i) \subseteq REM_j(R_k)$, where $R_2 \neq R_k$. $REM_j(R_2)$ is non-empty when $REM_j(R_1)$ is removed. $A \in REM_j(R_1) \cap REM_j(R_2)$, because it could not have been a solitary node while the remainders of R_1 and R_2 are both non-empty. Therefore, $A \in REM_j(R_k)$. Pick a node of G as a root and orient G such that the remainder of any child node was removed before the remainder of its parent. In this orientation of G, R_k is the parent of R_1.

R_2 cannot be in the subtree of G headed by R_1. The path from R_1 to R_2 must go through R_k. There is a shorter path from R_k to R_2 than from R_1 to R_2. Since $A \in R_2 \cap R_k$, and by the minimum distance assumption for R_1 and R_2, there is an A-path from R_k to R_2 in G. The edge (R_1, R_k) has A in its label. We conclude there is an A-path from R_1 to R_2, a contradiction. G must be a join graph, and hence a join tree for **R**.

Lemma 13.4 Let **R** be a connected database scheme. If **R** has a join tree, then R has a full-reducer.

Before proceeding with the proof of Lemma 13.4, we need some notation.

If $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ is a connected database scheme, and G is a join tree for **R**, let G_i represent G considered as an oriented tree with root R_i, $1 \leq i \leq p$. Let $d = \{r_1(R_1), r_2(R_2), \ldots, r_p(R_p)\}$ be a database over **R**. Consider a semijoin program $SP = sj_1, sj_2, \ldots, sj_k$ over d. Let j be a number between 1 and k. SP_j denotes the prefix $sj_1, sj_2, \ldots, sj_j$ of SP, which itself is a semijoin

program. SP_0 is the semijoin program with no steps. For $R_\ell \in \mathbf{R}$, j is a *completion point for* R_ℓ *in* SP *relative to* G_i if

1. for every child R of R_ℓ in G_i, if r is the relation on R and r_ℓ is the relation on r_ℓ, SP_j contains a step $r_\ell \leftarrow r_\ell \ltimes r$, and
2. for no $j' < j$ does condition 1 hold.

That is, the completion point for R_ℓ is the step in SP at which the relation for R_ℓ has been semijoined with all the relations for children of R_ℓ. If j is the completion point (should one exist) for R_ℓ in SP relative to G_i, we write $CP_i(R_\ell) = j$. If R_ℓ has no completion point in SP relative to G_i, we let $CP_i(R_\ell)$ be undefined. If R_ℓ is a leaf of G_i, let $CP_i(R_\ell) = 0$.

Example 13.29 Let $R = \{R_1,R_2,R_3,R_4,R_5\}$ be a database scheme where

$$R_1 = ABC \qquad R_3 = CDE \qquad R_5 = DJ.$$
$$R_2 = BCD \qquad R_4 = DI$$

Figure 13.24 contains a join tree G for R, which is oriented to be G_2. Let r_1, r_2, r_3, r_4, and r_5 be relations on R_1, R_2, R_3, R_4, and R_5, respectively. For the semijoin program $SP =$

1. $r_3 \leftarrow r_3 \ltimes r_4$
2. $r_3 \leftarrow r_3 \ltimes r_5$
3. $r_2 \leftarrow r_2 \ltimes r_1$
4. $r_2 \leftarrow r_2 \ltimes r_3$

$CP_2(R_3) = 2$ and $CP_2(R_2) = 4$. If we consider SP relative to G_3, shown in Figure 13.25, $CP_3(R_3)$ is undefined, since $r_3 \leftarrow r_3 \ltimes r_2$ does not occur in SP.

The semijoin program SP is *complete for* $\mathbf{R}$ *relative to* G_i if

1. $CP_i(R_\ell)$ is defined for every $R_\ell \in R$, and
2. if R is a child of R_ℓ in G_i, then $CP_i(R) < CP_i(R_\ell)$.

That is, by step $CP_i(R_\ell)$ in SP, the relation for R_ℓ, has been semijoined with the relations for all its children, whose relations, in turn, have been semijoined with the relations for all their children, and so forth.

Example 13.30 Let $\mathbf{R}$ and SP be as in Example 13.29. SP is complete for $\mathbf{R}$ relative to G_2, but not relative to G_1, G_3, or G_4.

For each G_i, there is at least one complete semijoin program for $\mathbf{R}$. For example, do a postorder traversal of G_i and when a node is visited, a semijoin of the relation for the parent of the node with the relation for the node is added

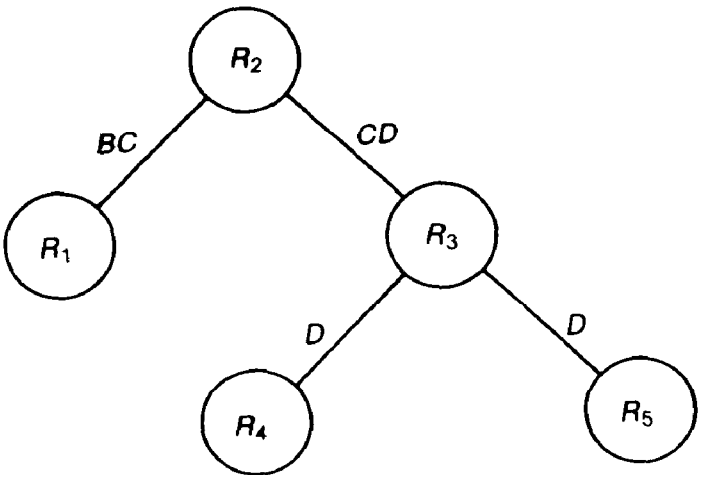

Figure 13.24

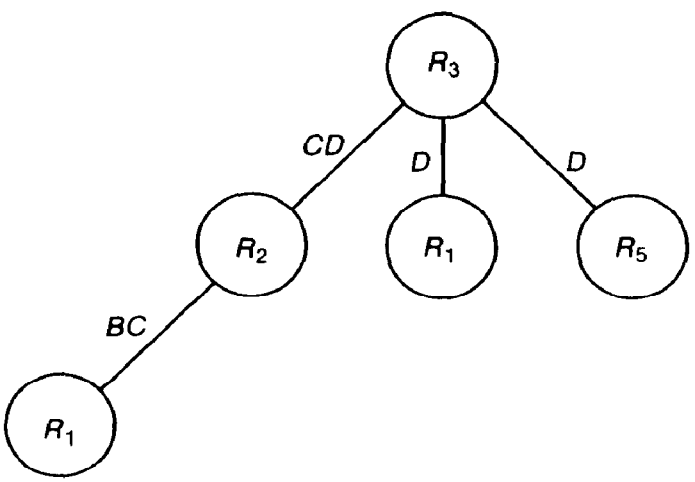

Figure 13.25

to the semijoin program. For each i, $1 \leq i \leq p$, let $SP(i)$ denote a minimal-length, complete semijoin program for **R** relative to G_i.

Example 13.31 Let **R** and SP be as in Example 13.29. If we use a postorder traversal of G_2, we get a complete semijoin program for **R**, $SP(2) =$

1. $r_2 \leftarrow r_2 \ltimes r_1$
2. $r_3 \leftarrow r_3 \ltimes r_4$
3. $r_3 \leftarrow r_3 \ltimes r_5$
4. $r_2 \leftarrow r_2 \ltimes r_3$.

$SP(2)$ is minimal-length.

Finally, some notation for the oriented trees $G_1, G_2, \ldots, G_p$. For $R_\ell \in \mathbf{R}$, $TREE_i(R_\ell)$ is the set of schemes in the subtree of G_i headed by R_ℓ. Note that $TREE_i(R_i) = \mathbf{R}$. The *extended scheme* of R_ℓ in G_i, $EX_i(R_\ell)$, is defined as

$$EX_i(R_\ell) = \cup \{R_j | R_j \in TREE_i(R_\ell)\}.$$

That is, $EX_i(R_\ell)$ is R_ℓ union all its descendents in G_i.

Proposition 13.5 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a connected database scheme and let G be a join tree for $\mathbf{R}$. Let $d = \{r_1(R_1), r_2(R_2), \ldots, r_p(R_p)\}$ be a database on $\mathbf{R}$. If SP is a complete semijoin program for $\mathbf{R}$ relative to G_i, then $SP(r_i,d) = FR(r_i,d)$.

Proof We prove a slightly stronger result. For $R_\ell \in R$, let d_ℓ be the subdatabase of d on the schemes in $TREE_i(R_\ell)$. Let $q = CP_i(R_\ell)$. We show that $SP_q(r_\ell,d) \subseteq FR(r_\ell,d_\ell)$. That is, at the completion point for R_ℓ in SP, r_ℓ is fully reduced relative to the relations for schemes in $TREE_i(R_\ell)$. Furthermore, for every tuple $t_\ell \in SP_q(r_\ell,d)$, there is a tuple $u_\ell \in \bowtie d_\ell$ such that $u_\ell(R_\ell) = t_\ell$. Note that the scheme for u_ℓ is $EX_i(R_\ell)$.

If R_ℓ is a leaf, the containment holds, for r_ℓ is fully reduced with respect to itself with no semijoins being applied. That is

$$SP_0(r_\ell,d) = FR(r_\ell, \{r_\ell\}).$$

Also $\bowtie d_\ell = r_\ell$, so for any tuple t_ℓ in r_ℓ, $\bowtie d_\ell$ contains a tuple $u_\ell (= t_\ell)$ such that $u_\ell(R_\ell) = t_\ell$.

Suppose now that R_ℓ is an interior node in G_i, with $q = CP_i(R_\ell)$ in SP. For notational convenience, assume $R_1, R_2, \ldots, R_m$ are the children of R_ℓ in G_i. We inductively assume the result holds for all of $R_1, R_2, \ldots, R_m$. Since SP is complete relative to G_i, $CP_i(R_j) < q$ for $1 \leq j \leq m$. At some point in SP_q, r_j was fully reduced relative to d_j. Furthermore, at that point, for every tuple $t_j \in t_j$, there is a tuple $u_j \in \bowtie d_j$ with $u_j(R_j) = t_j$. Can these properties be changed by semijoins subsequent to $CP_i(R_j)$? No. The only semijoins to worry about are those involving relations in d_j. Any semijoin that removes tuples

from r_j will not change the properties. Any semijoins that remove tuples from other relations in d_j must involve only relations in d_j, which cannot remove tuples used in $\bowtie d_j$. (Why?)

Let t_ℓ be a tuple in $SP_q(r_\ell,d)$. We must exhibit a tuple u_ℓ in $\bowtie d_\ell$ such that $u_\ell(R_\ell) = t_\ell$. Since q is the completion point for R_ℓ, r_ℓ has been semijoined with all of $r_1, r_2, \ldots, r_m$ in SP_q. Each r_j, $1 \leq j \leq m$ must contain a tuple t_j that joins with t_ℓ. (Again, semijoins subsequent to $r_\ell \leftarrow r_\ell \ltimes r_j$ cannot change this fact.) In turn, for each t_j, $\bowtie d_j$ contains a tuple u_j with $u_j(R_j) = t_j$. We claim we can form u_ℓ by $t_\ell \bowtie u_1 \bowtie u_2 \bowtie \cdots \bowtie u_m$.

We must show that $t_\ell, u_1, u_2, \ldots, u_m$ are joinable. To show that t_ℓ joins with u_j, $1 \leq j \leq m$, we show that $R_\ell \cap EX_i(R_j) \subseteq R_j$. Note that $EX_i(R_j)$ is the scheme of u_j. If $A \in EX_i(R_j)$, then $A \in R$ for some $R \in TREE_i(R_j)$. If $A \in R$, then there is an A-path from R to R_ℓ in G_i. This path necessarily passes through R_j, so $A \in R_j$. A similar argument shows that for $1 \leq j_1 < j_2 \leq m$, $R_\ell \supseteq EX_i(R_{j_1}) \cap EX_i(R_{j_2})$, so u_{j_1} and u_{j_2} only overlap in R_ℓ. Since u_{j_1} and u_{j_2} both agree with t_ℓ on R, they agree with each other. Since $t_\ell, u_1, u_2, \ldots, u_m$ agree pairwise, they are joinable (see Exercise 13.3). If u_ℓ is the result of joining $t_\ell, u_1, u_2, \ldots, u_m$, obviously $u_\ell(R_\ell) = t_\ell$. We conclude $SP_p(r_\ell,d) \subseteq FR(r_\ell,d_\ell)$.

To conclude, we have, in particular, that $SP(r_i,d) = SP(r_i,d_i) \subseteq FR(r_i,d)$. Since it is always the case that $SP(r_i,d) \supseteq FR(r_i,d)$, we have $SP(r_i,d) = FR(r_i,d)$,

Proof of Lemma 13.4 Let G be a join tree for $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$. Let $SP(1)$ be a minimal length, complete semijoin program for R relative to G_1. Let $\overline{SP}(1)$ be the semijoin program obtained from $SP(1)$ by reversing the order of the steps and changing each step $r_i \leftarrow r_i \ltimes r_j$ to $r_j \leftarrow r_j \ltimes r_i$. We leave it to the reader to show that the semijoin program SP equal to $SP(1)$ followed by $\overline{SP}(1)$ is complete for $\mathbf{R}$ relative to any G_ℓ, $1 \leq \ell \leq p$ (see Exercise 13.28). Note that SP has $2p - 2$ steps; this number is necessary (see Exercise 13.29). By Proposition 13.5, $SP(r_\ell,d) = FR(r_\ell,d)$, $1 \leq \ell \leq p$, so SP is a full-reducer for $\mathbf{R}$.

Lemma 13.5 Let $\mathbf{R}$ be a connected database scheme. If $\mathbf{R}$ has a full reducer SP, then PC implies TC for $\mathbf{R}$.

Proof We show the contrapositive. Let d be a database on $\mathbf{R}$ that is PC but not TC. Any semijoin program SP for $\mathbf{R}$ leaves d unchanged, so SP cannot be a full reducer for $\mathbf{R}$.

For the next lemma, we need some additional concepts for hypergraphs. Let $H = (\mathfrak{N}, \mathcal{E})$ be a connected hypergraph. An edge F in $\mathcal{E}$ is a *bottleneck* for H if $\mathcal{E} - \{F\}$ can be partitioned into two non-empty sets $\mathcal{E}_1$ and $\mathcal{E}_2$ such that for any $E_1 \in \mathcal{E}_1$ and $E_2 \in \mathcal{E}_2$, $E_1 \cap E_2 \subseteq F$. Removal of the nodes in F would disconnect H. Also, if $E_1, E_2, \ldots, E_k$ is a path in H from E_1 in $\mathcal{E}_1$ to E_k in $\mathcal{E}_2$, then for some i, $1 \leq i < k$, $E_i \cap E_{i+1} \subseteq F$. Therefore, for an edge F not to be a bottleneck, every pair of edges in $\mathcal{E} - \{F\}$ must be connected by a path that avoids F: no two consecutive edges in the path have an intersection that lies entirely within F.

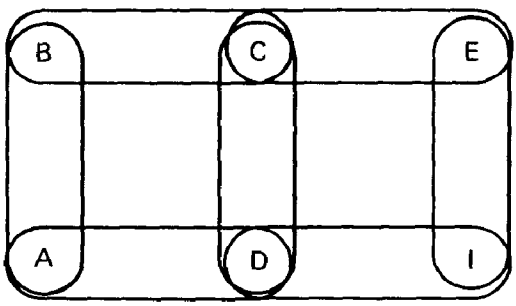

Figure 13.26

If F is a bottleneck to H relative to the sets $\mathcal{E}_1$ and $\mathcal{E}_2$, then the hypergraphs defined by $\mathcal{E}_1 \cup \{F\}$ and $\mathcal{E}_2 \cup \{F\}$ must be closed relative to H. Moreover, if H is cyclic, at least one of $\mathcal{E}_1 \cup \{F\}$ and $\mathcal{E}_2 \cup \{F\}$ is cyclic (see Exercise 13.30).

Lemma 13.6 Let **R** be a connected database scheme. If PC implies TC for **R**, then **R** is acyclic.

Proof We show the contrapositive: If **R** is cyclic then there exists a PC database d on **R** that is not TC. Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a smallest counterexample to the contrapositive. **R** is cyclic, but every PC database d on **R** is TC. Let p be minimum among all such counterexamples, and let the number of attributes in **R** be minimum relative to p. Graham reduction leaves **R** unchanged. If Graham reduction changed **R** to **R**′, then **R**′ is smaller than **R** in attributes or schemes, and is not a counterexample. **R**′ thus has a PC database d' that is not TC. Database d' can be extended to a database d on **R** that is not TC, by Exercise 13.31.

Considering **R** as a hypergraph, since Graham reduction does not apply, no edge of **R** contains a solitary node, nor is that edge contained in another edge. **R** cannot contain a bottleneck. Suppose R_i is a bottleneck, and, for notational convenience, $\{R_1, R_2, \ldots, R_{i-1}\}$ and $\{R_{i+1}, R_{i+2}, \ldots, R_p\}$ are two sets of edges that R_i separates. Both $R_1 = \{R_1, R_2, \ldots, R_i\}$ and $R_2 = \{R_i, R_{i+1}, \ldots, R_p\}$ must define closed subhypergraphs of **R**, at least one of which is cyclic. Say $\mathbf{R}_1$ is cyclic. $\mathbf{R}_1$ is smaller than **R** in number of schemes, so there is a database d_1 on $\mathbf{R}_1$ that is PC and not TC. Database d_1 can be extended into a PC database d on R by adding relations on $R_{i+1}, R_{i+2}, \ldots, R_p$ (see Exercise 13.32). Database d is not TC. (Why?) We have a contradiction to the definition of **R**, so **R** must have no bottleneck.

We are now ready to construct a database on **R** that is PC but not TC. Let $A_1, A_2, \ldots, A_n$ be the attributes in R_1, and let $A_{n+1}, A_{n+2}, \ldots, A_q$ be the rest of the attributes in **R**. We construct a relation $r(A_1\, A_2 \cdots A_q)$ with n tuples $t_1, t_2, \ldots, t_n$, defined as

$$t_i(A_j) = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \text{ and } j \leq n \\ i \text{ if } n < j \leq q. \end{cases}$$

Figure 13.27 shows relation r.

	$r(A_1$	A_2	...	A_i	...	A_n	A_{n+1}	A_{n+2}	...	$A_q)$
t_1	1	0	...	0	...	0	1	1	...	1
t_2	0	1	...	0	...	0	2	2	...	2
⋮					⋮					
t_i	0	0	...	1	...	0	i	i	...	i
⋮					⋮					
t_n	0	0	...	0	...	1	n	n	...	n

Figure 13.27

Let $r_i = \pi_{R_i}(r)$ for $1 \leq i \leq p$. We claim that $r_2 \bowtie r_3 \bowtie \cdots \bowtie r_p = r$. Any two schemes in $\{R_2, R_3, \ldots, R_p\}$ are connected by a path that avoids R_1, since R_1 is not a bottleneck. Hence, any tuple $t \in r_2 \bowtie r_3 \bowtie \cdots \bowtie r_p$ must have the same value on each of $A_{n+1}, A_{n+2}, \ldots, A_q$. Suppose the value of t is i on all of these attributes. We show that $t = t_i$. Consider any scheme R_j, $2 \leq j \leq p$, that contains one or more attributes from among $A_1, A_2, \ldots, A_p$. Since $R_1 \not\supseteq R_j$, R_j also contains at least one attribute from among $A_{n+1}, A_{n+2}, \ldots, A_q$, say A_ℓ. If u_j is the tuple from r_j that contributed to t, then $u_j(A) = i$. It must be that $u_j = t_i(R_j)$. We conclude that t agrees with t_i wherever t is de-

fined. Since no attribute in R_1 is solitary, t must be defined on all of $A_1 A_2 \cdots A_q$. We see that $r \supseteq r_2 \bowtie r_3 \bowtie \cdots \bowtie r_p$. The other containment is a direct property of project-join mappings, so $r = r_2 \bowtie r_3 \bowtie \cdots \bowtie r_p$.

Since $r_1, r_2, \ldots, r_p$ are all projections of the same relation, they are TC and hence PC. Let $s_1 = r_1 \cup \{\langle 0\,0 \ldots 0\rangle\}$. That is, s_1 is r_1 plus the tuple of all 0's. We claim that $s_1, r_2, r_3, \ldots, r_p$ are PC. For each R_j, $2 \leq j \leq p$,

$$\pi_{S_1 \cap S_j}(s_1) = \pi_{S_1 \cap S_j}(r_1)$$

since $S_1 \cap S_j \neq S_j$. The projection, in both cases, contains the tuple of all 0's plus every tuple with one 1 and 0's elsewhere. Thus, s_1 is consistent with each of $r_2, r_3, \ldots, r_p$, which are already known to be consistent among themselves.

The database $d = \{s_1, r_2, r_3, \ldots, r_p\}$ is a PC database on **R**. However, d cannot be TC, since $r_2 \bowtie r_3 \bowtie \cdots \bowtie r_p = r$ and s_1 and r do not join completely. **R** cannot be a counterexample, and the lemma is proved.

Lemma 13.7 Let **R** be a connected database scheme. If **R** has a join tree then **R** has the running intersection property.

Proof Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ and let G be a join tree for **R**. Assume $R_1, R_2, \ldots, R_p$ are in preorder according to G_1. It follows that if R_j is an ancestor of R_k in G_1, then $j < k$. Consider any R_i for $2 \leq i \leq p$. One of $R_1, R_2, \ldots, R_{i-1}$ is the parent of R_i in G_1. Let it be R_j. None of $R_1, R_2, \ldots, R_{i-1}$ is a descendent of R_i. Let A be any attribute in $(R_1 R_2 \cdots R_{i-1}) \cap R_i$. There must be an A-path from R_i to one of $R_1, R_2, \ldots, R_{i-1}$, and this A-path necessarily passes through R_j. Hence $R_j \supseteq (R_1 R_2 \cdots R_{i-1}) \cap R_i$ and so **R** has the running intersection property.

Lemma 13.8 Let **R** be a connected database scheme. If **R** has the running intersection property then **R** has the increasing join property.

Proof Let $R_1, R_2, \ldots, R_p$ be an ordering of the schemes in R such that for $2 \leq i \leq p$, $(R_1 R_2 \cdots R_{i-1}) \cap R_i \subseteq R_j$ for some $1 \leq j < i$. Let $d = \{r_1(R_1), r_2(R_2), \ldots, r_p(R_p)\}$ be a PC database on **R**. Let JP be the join plan corresponding to the parenthesized join expression

$$(\cdots((r_1 \bowtie r_2) \bowtie r_3) \cdots \bowtie r_p).$$

We show inductively that

$$\pi_{R_j}(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_i) = r_j \text{ for } 1 \leq j \leq i,$$

which means $r_1, r_2, \ldots, r_i$ are TC.

The basis is immediate. Since r_1 and r_2 are consistent, $\pi_{R_1}(r_1 \bowtie r_2) = r_1$ and $\pi_{R_2}(r_1 \bowtie r_2) = r_2$. Suppose the hypotheses are true for $i - 1$. Consider r_i. Let R_j be a scheme such that $j < i$

$$R_j \supseteq (R_1 R_2 \cdots R_{i-1}) \cap R_i = S.$$

Since $\pi_{R_j}(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_{i-1}) = r_j$, it follows that $\pi_S(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_{i-1}) = \pi_S(r_j)$. Since r_i is consistent with r_j, r_i joins completely with $\pi_S(r_j)$ and hence with $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_{i-1}$. Since $r_1, r_2, \ldots, r_{i-1}$ join completely, so do $r_1, r_2, \ldots, r_i$. It follows that

$$\pi_{R_i}(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_i) = r_i$$

and, more generally, that

$$\pi_{R_j}(r_1 \bowtie r_2 \bowtie \cdots \bowtie r_i) = r_j, \text{ for } 1 \leq j \leq i.$$

Since the joins

$$r_1 \bowtie r_2 \bowtie \cdots \bowtie r_i \text{ for } 2 \leq i \leq p$$

are exactly the joins corresponding to the interior nodes of join plan JP, we see that JP is a monotone join plan. Thus, **R** has the increasing join property.

Lemma 13.9 Let **R** be a connected database scheme. If **R** has the increasing join property, then PC implies TC for databases for **R**.

Proof Let JP be a monotone join plan for **R** and let d be a PC database on **R**. JP gives a method to join all the relations in d such that no tuples are lost along the way. Therefore, d is TC.

Lemma 13.10 Let **R** be a connected database scheme. If **R** has a join tree, then $MVD(R) \models *[\mathbf{R}]$.

Proof Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ and let G be a join tree for **R**. Recall that G_1 is G viewed as an oriented tree with R_1 as the root. Choose any R_i, $2 \leq$

$i \leq p$, and let R_j be its parent in G_1. Let $S_i = EX_1(R_i)$. That is, S_i is the union of all the schemes in the subtree headed by $\mathbf{R}_i$. Let S_j be the union of all the rest of the schemes in $\mathbf{R}$. We claim $S_i \cap S_j = R_i \cap R_j$.

$S_i \cap S_j \supseteq R_i \cap R_j$ is immediate because $S_i \supseteq R_i$ and $S_j \supseteq R_j$. To see the other inclusion, the presence of any A in $S_i \cap S_j$ implies an A-path through R_i and R_j, so that $A \in R_i \cap R_j$. We thus have the equality.

From the remarks after Lemma 13.1, we now know that $*[\mathbf{R}] \models *[S_i, S_j]$. In MVD notation, $*[S_i, S_j]$ is $R_i \cap R_j \twoheadrightarrow S_i | S_j$. We use G_1 as a plan for chasing the tableau $T_{\mathbf{R}}$ so as to yield the row of all distinguished variables. We show, recursively, that for each R_i, $1 \leq i \leq p$, we can derive a row that is a's (distinguished variables) on exactly $EX_1(R_i)$ in computing $chase_{MVD(\mathbf{R})}(T_{\mathbf{R}})$. Since $EX_1(R_1) = R_1 R_2 \cdots R_p$, establishing this result proves the lemma.

If R_i is a leaf in G_1, then $EX_1(R_i) = R_i$ and we have a row distinguished exactly on R_i in $T_{\mathbf{R}}$ initially. If R_i is an interior node in G_1, let $Q_1, Q_2, \ldots, Q_k$ be its children. Assume that $T_{\mathbf{R}}$ has been chased under $MVD(\mathbf{R})$ to a tableau $T'_{\mathbf{R}}$ that has a row w_j distinguished on $EX_1(Q_j)$ for $1 \leq j \leq k$. Let v be the row that is distinguished on R_i. For each Q_j, we have that $R_i \cap Q_j \twoheadrightarrow EX_1(Q_j)$ by the initial paragraphs of this proof. Applying $R_i \cap Q_j \twoheadrightarrow EX_1(Q_j)$ to v and w_j for $1 \leq j \leq k$ will transform v into the row distinguished on exactly $R_i \cup EX_1(Q_1) \cup EX_1(Q_2) \cdots \cup EX_1(Q_k)$. That is, v is distinguished $EX_1(R_i)$. Note that the distinguished variables that $R_i \cap Q_j \twoheadrightarrow EX_1(Q_j)$ adds to v are not removed by $R_i \cap Q_\ell \twoheadrightarrow EX_1(Q_\ell)$, $j \neq \ell$, since $R_i \supseteq EX_1(Q_j) \cap EX_1(Q_\ell)$.

We shall shortly be looking at tight decompositions of a scheme $\mathbf{U}$ relative to a set of MVDs, where the set is $MVD(\mathbf{R})$ for a database scheme $\mathbf{R}$ over $\mathbf{U}$. If (S_1, S_2) is a tight decomposition of $\mathbf{U}$ relative to $MVD(\mathbf{R})$, there can be no MVD $*[S'_1, S'_2]$ in $MVD(\mathbf{R})$ such that $S'_1 \cap S'_2$ is properly contained in $S_1 \cap S_2$. Since $MVD(\mathbf{R}) \models *[S_1, S_2]$, $*[S_1, S_2]$ must be in $MVD(\mathbf{R})$. Thus $\mathbf{R}$ can be partitioned into $\mathbf{R}_1$ and $\mathbf{R}_2$ such that $\cup\mathbf{R}_1 = S_1$ and $\cup\mathbf{R}_2 = S_2$. For the proof of the next lemma, we need the following proposition.

Proposition 13.6 Let $\mathbf{R}$ be a reduced, connected, acyclic database scheme over $\mathbf{U}$. Suppose (S_1, S_2) is a tight decomposition of $\mathbf{U}$ relative to $MVD(\mathbf{R})$. The set $X = S_1 \cap S_2$ is an articulation set for $\mathbf{R}$.

Proof Let $\mathbf{R}$ be partitioned into $\mathbf{R}_1$ and $\mathbf{R}_2$ such that $\cup\mathbf{R}_1 = S_1$ and $\cup\mathbf{R}_2 = S_2$. The removal of the attributes in X surely disconnects $\mathbf{R}$, so the only way X can fail to be an articulation set is by not being the intersection of two edges in $\mathbf{R}$ (treating $\mathbf{R}$ as a hypergraph).

Assume that $\mathbf{R}_1$ has no edge containing X. We assume that every pair of edges in $\mathbf{R}_1$ is connected by a path in $\mathbf{R}_1$ that avoids X. If not, pick an edge R in $\mathbf{R}_1$. Move all the edges in $\mathbf{R}_1$ that are not connected to R, by a path that avoids X, to $\mathbf{R}_2$. By the minimality of $S_1 \cap S_2$, the movement of these edges preserves the property that $(\cup\mathbf{R}_1) \cap (\cup\mathbf{R}_2) = X$.

We show a contradiction by showing that Graham reduction can never succeed on $\mathbf{R}$. In particular, we show that the nodes in X never get removed. Initially, any $A \in X$ is contained in an edge from $\mathbf{R}_1$ and an edge from $\mathbf{R}_2$, so it is not solitary. We show that Graham reduction preserves this property.

First we look at $\mathbf{R}_1$. Let $Y_1, Y_2, \ldots, Y_m$ be the maximal intersections of edges in $\mathbf{R}_1$ with X. That is, for each Y_i, $1 \leq i \leq m$, $\mathbf{R}_1$ contains an edge R_i such that $R_i \cap X = Y_i$ and for no other edge $R_j \in \mathbf{R}_1$ does $R_j \cap X$ properly contain $R_i \cap X$. For each Y_i, $1 \leq i \leq m$, there is a Y_j, $i \neq j$, with edges $R_i \supseteq Y_i$ and $R_j \supseteq Y_j$ in $\mathbf{R}_1$ such that R_i and R_j are connected by a path that avoids X. Let the path be $R_i = S_1, S_2, \ldots, S_k = R_j$. Node removal preserves this path, since no node in the intersection of successive edges can be solitary, and none of the nodes in Y_i or Y_j is solitary. If some edge S_ℓ in $S_1, S_2, \ldots, S_k$ is removed because it is contained in another edge Q, Q must be in $\mathbf{R}_1$, since S_ℓ contains a node not in X. If Q is not in the path, replace S_ℓ by Q in the path. All the properties of the path are preserved. If Q is already in the path, remove the portion of the path from S_ℓ to Q. Q cannot contain both $S_1(=R_1)$ and $S_m(=R_j)$ by the maximality of Y_i and Y_j. In this case also, all the properties of the path are preserved.

Thus, at every point in Graham reductions, Y_i is contained by an edge $R_i \subseteq \mathbf{R}_1$, and every node in X is contained in some Y_i, $1 \leq i \leq m$.

Consider the maximal intersections $Z_1, Z_2, \ldots, Z_n$ of edges in $\mathbf{R}_2$ with X. If for some Z_i, $1 \leq i \leq n$, there is a Z_j, $i \neq j$, with edges $R_i \supseteq Z_i$ and $R_j \supseteq Z_j$ with R_i and R_j connected by a path in $\mathbf{R}_2$ that avoids X, then there will always be an edge in $\mathbf{R}_2$ containing Z_i, by the argument above. If there is no such path, consider any $R_i \supseteq Z_i$ in $\mathbf{R}_2$. If R_i had nodes outside of X, then $\{R_i\}$ and $\mathbf{R} - \{R_i\}$ could have been used to form a decomposition of $\mathbf{U}$ that was "tighter" than (S_1, S_2). Hence $R_i \subseteq X$, and so $R_i = Z_i$. R_i cannot be contained in any other edge in $\mathbf{R}$, since $\mathbf{R}$ is reduced. Further, every node in R_i is contained in some edge of $\mathbf{R}_1$, so R_1 can never be reduced by node removal.

In either case, there is always an edge R_i in $\mathbf{R}_2$ containing Z_i during Graham reduction. Since every node in X is in some Y_i, $1 \leq i \leq m$, and some Z_j, $1 \leq j \leq n$, during Graham reduction, Graham reduction fails on $\mathbf{R}$, a contradiction. Both $\mathbf{R}_1$ and $\mathbf{R}_2$ must contain edges containing X, so X is an articulation set.

Lemma 13.11 Let $\mathbf{R}$ be a reduced, connected, acyclic database scheme. $\mathbf{R}$ is a unique 4NF decomposition.

Proof Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ and let $\mathbf{U} = R_1 R_2 \cdots R_p$. We show that $\mathbf{R}$ is a unique decomposition for $\mathbf{U}$ under $MVD(\mathbf{R})$. We need only consider the case where $p \geq 2$. Since $\mathbf{R}$ must have articulation sets, there must be MVDs in $MVD(\mathbf{R})$ that can be used to decompose $\mathbf{U}$. By the discussion before Proposition 13.6, if (S_1, S_2) is a tight decomposition of $\mathbf{U}$ relative to $MVD(\mathbf{R})$, then $\mathbf{R}$ can be partitioned into $\mathbf{R}_1$ and $\mathbf{R}_2$ with $\cup\mathbf{R}_1 = S_1$ and $\cup\mathbf{R}_2 = S_2$. By Proposition 13.6, each of $\mathbf{R}_1$ and $\mathbf{R}_2$ includes an edge that contains $S_1 \cap S_2$. Thus, $\mathbf{R}_1$ and $\mathbf{R}_2$ are closed relative to $\mathbf{R}$.

Let M_i be the set of MVDs that $MVD(\mathbf{R})$ induces on S_i, $1 \leq i \leq 2$. We claim that M_i is equivalent to $MVD(\mathbf{R}_i)$. This claim is sufficient to prove the lemma. Since $\mathbf{R}_1$ and $\mathbf{R}_2$ are closed relative to $\mathbf{R}$, they are both reduced, connected, and acyclic. Every $R \in \mathbf{R}$ is in either S_1 or S_2. If we inductively assume that $\mathbf{R}_1$ is a unique decomposition for S_1 relative to $MVD(\mathbf{R}_1)$, and $\mathbf{R}_2$ is a unique decomposition for S_2 relative to $MVD(\mathbf{R}_2)$, then the claim allows us to conclude that $\mathbf{R}$ is a unique decomposition of $\mathbf{U}$ relative to $MVD(\mathbf{R})$.

Consider S_1 and $\mathbf{R}_1$. The claim follows from comparing chasing tableaux on S_1 under $MVD(\mathbf{R}_1)$ and chasing tableaux on $\mathbf{U}$ under $MVD(\mathbf{R})$. For a tableau T over S_1, let T *extended to* $\mathbf{U}$, denoted $T^{\mathbf{U}}$, be obtained by padding each row in T with new nondistinguished variables on $\mathbf{U} - S_1$. If w is a row in T, let $w^{\mathbf{U}}$ be the corresponding row in $T^{\mathbf{U}}$.

Suppose we are testing whether some MVD on S_1 is implied by $*[\mathbf{R}_1]$. Let T be the tableau for the MVD. Whatever changes made to T using $*[\mathbf{R}_1]$ can be mimicked on $T^{\mathbf{U}}$ using $*[\mathbf{R}]$ in such a way that $T^{\mathbf{U}}$ restricted to S_1 equals T. Suppose $\mathbf{R}_1 = \{R_1, R_2, \ldots, R_q\}$ and $\mathbf{R}_2 = \{R_{q+1}, R_{q+2}, \ldots, R_p\}$. Suppose the J-rule for $*[\mathbf{R}_1]$ is used on rows $w_1, w_2, \ldots, w_q$ in T to yield row v. For convenience, assume R_q is an edge in $\mathbf{R}_1$ containing $S_1 \cap S_2$. We can apply $*[\mathbf{R}]$ to rows $w_1^{\mathbf{U}}, w_2^{\mathbf{U}}, \ldots, w_q^{\mathbf{U}}, w_q^{\mathbf{U}}, \ldots, w_q^{\mathbf{U}}$ to yield a row $v^{\mathbf{U}}$ such that $v^{\mathbf{U}}(S_1) = v$. Therefore, if we ever arrive at the row of all distinguished variables in T, there is a row in $T^{\mathbf{U}}$ that is distinguished on S_1. Hence, any MVD on S_1 implied by $*[\mathbf{R}_1]$ is an embedded MVD on $\mathbf{U}$ implied by $*[\mathbf{R}]$. That is, the MVD is in M_1, so $MVD(\mathbf{R}_1) \subseteq M_1$.

To show the other containment, we use the following property of the chase. If the chase of a tableau is being computed under a single JD, then any row derived during the chase can be derived directly from the original rows in the tableau (see Exercise 13.33). Suppose we use the chase on a tableau T over $\mathbf{U}$ to show that $*[\mathbf{R}]$ implies an MVD embedded in S_1. The chase must have produced a row w that was distinguished on all of S_1. That row can be produced in one step from the original rows in T, by the property of the chase given above. It follows that if T' is T restricted to S_1, then the row of all distinguished variables can be produced in T' by one application of the J-rule for $*[\mathbf{R}_1]$. We conclude $M_1 \subseteq MVD(\mathbf{R}_1)$, so $M_1 = MVD(\mathbf{R}_1)$. By symmetry, $M_2 = MVD(\mathbf{R}_2)$, so the claim is established and the lemma is proved.

For the final lemma of this chapter, we need two more propositions and some definitions.

Proposition 13.7 Let M be a set of MVDs over $\mathbf{U}$. Let $R \subseteq S \subseteq \mathbf{U}$. If M implies a nontrivial MVD $X \twoheadrightarrow Y$ embedded in R, then M implies some nontrivial MVD $X \twoheadrightarrow Z$ embedded in S.

Proof Left to the reader as Exercise 13.34.

The next lemma states that if $\mathbf{R}$ is a unique decomposition, then $\mathbf{R}$ is acyclic. We shall represent tight 4NF decompositions by trees.

Definition 13.24 Let M be a set of MVDs over $\mathbf{U}$. A *decomposition tree* for $\mathbf{U}$ over M is a rooted binary tree with the following properties:

1. The nodes in G are labeled with subsets of $\mathbf{U}$.
2. The root of G is labeled with $\mathbf{U}$.
3. If a node labeled R has children labeled R_1 and R_2, then (R_1,R_2) is a tight 4NF decomposition of R relative to M.
4. If R labels a leaf of G, R is in 4NF relative to M.

Clearly, if G is a decomposition tree for $\mathbf{U}$ under M, then the labels of the leaves of G form a tight 4NF decomposition for $\mathbf{U}$ under M. If v is an interior node in G, we let $INT(v)$ be the intersection of the schemes of the children of v.

Example 13.31 Let $\mathbf{U} = A\ B\ C\ D\ E$ and let $M = \{A \twoheadrightarrow B,\ D \twoheadrightarrow E\}$. Figure 13.28 shows a decomposition tree G for $\mathbf{U}$ under M. In G, $INT(v_1) = A$ and $INT(v_3) = D$.

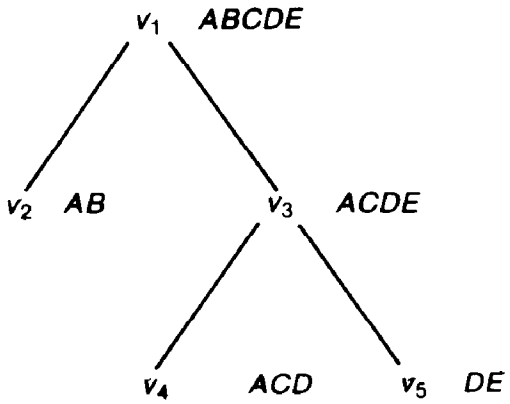

Figure 13.28

By Proposition 13.7, if v is an ancestor of w in a decomposition tree G, we cannot have $INT(v) \supsetneq INT(w)$, or else the decomposition at v was not tight. Further, the labels of nodes must be nonincreasing along every root-leaf path. That is, if v is an ancestor of w, the label of v contains the label of w. If x and y are nodes in G with labels R and S, but neither node is the ancestor of the other, then $R \not\supseteq S$. Let z be an ancestor of both x and y. $INT(z)$ must contain $R \cap S$. If $R \supseteq S$, then $R \cap S = S$. However, $S \supseteq INT(z)$, and so the decomposition at Z was trivial. It follows that no two leaves in G can be labeled with the same scheme.

Lemma 13.12 Let **R** be a connected database scheme. If **R** is a unique decomposition, then **R** is acyclic.

Proof We show that no cyclic database scheme can be a unique decomposition. First consider the case where **R** consists of a single block. Suppose **R** is a unique decomposition for **U** under a set M of MVDs. Let G be a decomposition tree for **U** under M. Consider an interior node v of G at one level up from the leaves. Let R be the label of the left child of v and S be the label of the right child, with $INT(v) = Y \cap Z = X$. Both Y and Z are schemes from **R**, since they label leaves of G. By Proposition 13.7, there must be some nontrivial MVD $X' \twoheadrightarrow W$ on **U** implied by M with $X' \subseteq X$. Assume that no nontrivial MVD on **U** implied by M has a left side contained in X'.

Construct another decomposition tree G' for **U** under M by using $X' \twoheadrightarrow W$ to decompose **U** at the first step. The other decompositions in G' are arbitrary. Since **R** is a unique decomposition relative to M, the labels on leaves of G' are the same as G, namely, all the schemes in **R**. Let $\mathbf{R}_1$ be all the labels of leaves in the left subtree of the root. Let $\mathbf{R}_2$ be the corresponding set for the right subtree. $\mathbf{R}_1$ and $\mathbf{R}_2$ are disjoint by the remarks before this lemma. The removal of X' separates $\mathbf{R}_1$ from $\mathbf{R}_2$. Since Y and Z are in **R**, and $Y \cap Z = X \supseteq X'$, **R** has an articulation set. (Note that no edge may be contained in X.)

The remainder of the proof is left to the reader as Exercise 13.35. The strategy is to show that if **R** is cyclic and is a unique decomposition, but has an articulation set, then a smaller counterexample to the lemma could be found by breaking **R** at the articulation set.

13.3.4 Conclusions

We have seen several syntactic and operational characterizations for acyclic schemes. The exercises present more characterizations. One interesting class

of questions about cyclic schemes is how they may be transformed or altered to produce acyclic schemes. Some possibilities are merging schemes, adding attributes to relation schemes, deleting attributes from relation schemes, breaking the database scheme into acyclic components, and adding new relation schemes. Unfortunately, most of these modifications are NP-complete if the minimum modification is sought. Another area for further work is how to exploit local acyclicity in a database scheme that is globally cyclic. Also, much work is going on in determining how data dependencies ameliorate the effects of cyclicity.

13.4 EXERCISES

13.1 Consider the database scheme $\mathbf{R}_a = \{ABC, BCE, CDE\}$. Give a database $d(\mathbf{R}_a) = \{r_1(ABC), r_2(BCD), r_3(CDE)\}$ and two full-reducers SP_1 and SP_2 such that SP_1 is beneficial to apply to d before computing $r_1 \bowtie r_2 \bowtie r_3$, but S_2 is not. Assume that all domain values have a transmission cost of 1, the cost of $r_i \ltimes r_j$ is the cost of transmitting r_j's projection on r_i's scheme, and the join is to be computed at the site of r_1.

13.2 Can the assignment $r \leftarrow r \ltimes s$ change r if the schemes of r and s do not intersect?

13.3 Show that a database d where every relation has a single tuple can always be fully reduced with semijoins. (Alternatively, PC implies TC for d.)

13.4 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a database scheme. Suppose $*[\mathbf{R}] \models *[S_1, S_2]$. Show that there is a pair of relation schemes S_1', S_2' such that for some function

$$f: \{1, 2, \ldots, p\} \rightarrow \{1, 2\}$$

we have

$$S_i' = \bigcup_{f(j)=i} R_j,\ i = 1, 2$$

and $*[S_1', S_2'] \models *[S_1, S_2]$.

13.5* If $\mathbf{R}$ has p schemes, how big can $MVD(\mathbf{R})$ be?

13.6 For the database schemes $\mathbf{R}_a$ and $\mathbf{R}_c$ in Example 13.4, verify that $MVD(\mathbf{R}_a) = *[\mathbf{R}_a]$ while $MVD(\mathbf{R}_c) \not\models *[\mathbf{R}_c]$.

13.7* Show that if $MVD(\mathbf{R}) \models *[\mathbf{R}]$, then there is a set of MVDs M equivalent to $MVD(\mathbf{R})$ with no more elements than schemes in $\mathbf{R}$.

13.8 Let $\mathbf{R}$ be a database scheme over $\mathbf{U}$. What is the time complexity of the obvious algorithm to test if a relation $r(\mathbf{U})$ satisfies *[R]? (The obvious algorithm is computing $m_{\mathbf{R}}(r)$.) How fast can satisfaction of *[R] be tested if $MVD(\mathbf{R}) \models$ *[R]? (Use Exercise 13.7.)

13.9 Give a "non-tight" decomposition of $A\ B\ C\ D\ E\ I$ under the MVDs $\{BC \twoheadrightarrow E,\ CD \twoheadrightarrow I\}$.

13.10 Show that if $\mathbf{R}$ is a unique decomposition of $\mathbf{U}$, then $MVD(\mathbf{R})$ uniquely decomposes $\mathbf{U}$.

13.11 Consider the database scheme $\mathbf{R}_c = \{ABC, BCD, CE, DE\}$ from the examples.
(a) Show that $\mathbf{R}_c$ is not a unique decomposition.
(b) Show two tight 4NF database schemes for $A\ B\ C\ D\ E$ under $MVD(\mathbf{R}_c)$.

13.12 Prove that if a database d is TC it must also be PC.

13.13 Show that for any $n \geq 3$ there is a database d of n relations such that any $n - 1$ relations join completely but d is not TC.

13.14 Verify that any way of computing $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$ using pairwise joins for the database in Figure 13.6 gives at least one intermediate result that is not the complete join of its child relations.

13.15* Let $\mathbf{R}$ be a database scheme such that for any join plan P for $\mathbf{R}$ there exists a database $d(\mathbf{R})$ such that $P(d)$ is not monotone. Show that there is a database $\hat{d}$ that is such that $P(\hat{d})$ is not monotone for *any* join plan for $\mathbf{R}$.

13.16 Let P be the join plan of Figure 13.9 and let d be the database in Figure 13.6. Verify that $P(d)$ is not monotone.

13.17 Show that the database $\mathbf{R}_c = \{ABC, BCE, CE, DE\}$ has no monotone join plan.

13.18 Can a database scheme have a join plan P where $P(d)$ is never monotone (excluding a database of empty relations)?

13.19 Enumerate all the reduced hypergraphs on five nodes (up to isomorphism).

13.20 Let H be a non-reduced cyclic hypergraph and let H' be its reduction. Show that $H'_{\mathfrak{M}}$ could be a block while $H_{\mathfrak{M}}$ is not, for some $\mathfrak{M}$. Can H have no blocks at all?

13.21 Determine whether each of the following database schemes are cyclic or acyclic.
(a) $\{ABC, CDE, AIE, ACE\}$
(b) $\{ABC, BCD, ACD, ABD\}$
(c) $\{AB, BD, CD, CE, DE\}$

13.22 Prove that a hypergraph is acyclic if and only if it is closed-acyclic.

13.23 Find join trees for the acyclic schemes in Exercise 13.21.

13.24 Show that the acyclic schemes in Exercise 13.21 have the running intersection property.

13.25 Show that the hypergraph for a database scheme $\mathbf{R}$ consists of a single connected component if and only if every join graph for $\mathbf{R}$ is connected.

13.26 Find a join tree for each acyclic database scheme in Exercise 13.21.

13.27 Show that if $H = (\mathfrak{N}, \mathcal{E})$ is an acyclic hypergraph, then so is $H_{\mathfrak{M}}$ for any $\mathfrak{M} \subseteq \mathfrak{N}$.

13.28 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a connected database scheme with join tree G. Let $SP(1)$ be a complete semijoin program for R relative to G_1. Show that $SP = SP(1); \overline{SP(1)}$ is a complete semijoin program for any G_ℓ, $1 \leq \ell \leq p$, where $\overline{SP}(1)$ is $SP(1)$ reversed and with each step $r_i \leftarrow r_i \ltimes r_j$ changed to $r_j \leftarrow r_j \ltimes r_i$.

13.29* Let $\mathbf{R}$ be a database scheme. Show that if $\mathbf{R}$ has a full reducer SP, then SP must have at least $2 \cdot |\mathbf{R}| - 2$ steps.

13.30 Let $H = (\mathfrak{N}, \mathcal{E})$ be a cyclic hypergraph. Let $F \in \mathcal{E}$ be a bottleneck for H relative to the partition $\mathcal{E}_1$, $\mathcal{E}_2$ of $\mathcal{E} - \{F\}$. Show that at least one of the hypergraphs defined by $\mathcal{E}_1 \cup \{F\}$ and $\mathcal{E}_2 \cup \{F\}$ is cyclic.

13.31 Let $\mathbf{R}$ be a connected database scheme and let $\mathbf{R}'$ be $\mathbf{R}$ after applying Graham reduction. Show that if $\mathbf{R}'$ has a PC database that is not TC, so does $\mathbf{R}$.

13.32 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a connected database scheme. Suppose R_i is a bottleneck for R relative to $\{R_1, R_2, \ldots, R_{i-1}\}$ and $\{R_{i+1}, R_{i+2}, \ldots, R_p\}$. Let d' be a PC database on $\{R_1, R_2, \ldots, R_i\}$. Show that d' can be extended to a PC database on $\mathbf{R}$ by adding relations on $R_{i+1}, R_{i+2}, \ldots, R_p$.

13.33 Consider taking the chase of a tableau T under a single JD $*[R_1, R_2, \ldots, R_p]$. Let w be a row at any point in the chase. Show that for any R_i, $1 \leq i \leq p$, there is an original row v in the chase such that $w(R_i) = v(R_i)$.

13.34 Prove Proposition 13.7.

13.35 Complete the proof of Lemma 13.12.

13.36* Show that condition 10 of Theorem 13.2 implies one of the conditions 1-9.

Definition 13.25 Let $H = (\mathfrak{N}, \mathcal{E})$ be a hypergraph and let $P = E_1, E_2, \ldots, E_m$ be a path in H. Define

$$F_i = E_i \cap E_{i+1},\ 1 \leq i \leq m.$$

P is *chordless* if there is no edge E in $\mathcal{E}$ that contains $F_i \cup F_j \cup F_k$ for some $1 \leq i < j < k < m$. That is, no edge in $\mathcal{E}$ contains three intersections of adjacent edges in the path. P is a *cycle* if $E_1 = E_m$.

13.37 Let H be a hypergraph. Prove: H is acyclic if and only if H contains no chordless cycles of 3 or more edges (counting the first and last edge only once).

13.38 Give an example of an acyclic hypergraph with a cycle.

13.39* Let d be a database on scheme $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$. Show that if a full reducer exists for d, it must have at least $2p - 2$ semijoins.

Definition 13.26 Let d be a database on scheme $\mathbf{R}$. A semijoin program SP is a *maximal reducer* for d if for any state of d, after applying SP to d, no semijoin will further reduce d (although d need not be fully reduced after applying SP).

13.40* Show that if a database d on scheme $\mathbf{R}$ has no full reducer, then it has no maximal reducer.

13.41 Say a join plan JP is *sequential* if every right child in JP is a leaf. Show that a database scheme $\mathbf{R}$ has a monotone join plan if and only if it has a monotone, sequential join plan.

Definition 13.27 Let $H = (\mathfrak{N}, \mathcal{E})$ be a hypergraph. The *graph for* H, G_H, is an ordinary, undirected graph on the nodes in $\mathfrak{N}$ that contains an edge (A, B) exactly when A and B are contained in a single edge of $\mathcal{E}$.

Definition 13.28 Let G be an undirected graph. A *clique* of G is a subset of nodes of G such that every pair of nodes in the subset forms an edge in the graph. G is *chordal* if every cycle of 4 or more nodes has a *chord*: an edge in G connecting non-adjacent nodes in the cycle.

Definition 13.29 Let H be a hypergraph. H is *conformal* if every set of nodes $\mathfrak{M}$ that is a clique of G_H is contained in a hyperedge of H. H is *chordal* if it is *conformal* and G_H is *chordal*.

13.42 Prove that a hypergraph H is acyclic if and only if it is chordal.

Recall that for a set of MVDs M and a set of attributes X, $DEP(X)$ is the dependency basis of X.

Definition 13.30 Let M be a set of MVDs. Let X be a *key* of M if X is the left side of an MVD in M. Two keys X and Y in M are *conflict-free* if we can write $DEP(X)$ and $DEP(Y)$ as

$$DEP(X) = \{V_1, V_2, \ldots, V_k, X_1, X_2, \ldots, X_m, Z_1 Y_1 Y_2 \cdots Y_n\}$$

and

$$DEP(Y) = \{V_1, V_2, \ldots, V_k, Y_1, Y_2, \ldots, Y_n, Z_2 X_1 X_2 \cdots X_m\}$$

such that

1. $Z_1 X = Z_2 Y$,
2. $DEP(X) \cap DEP(Y) = \{V_1, V_2, \ldots, V_k\}$, and
3. $DEP(X \cap Y) \supseteq \{V_1, V_2, \ldots, V_k\}$.

M is *conflict-free* if every pair of keys in M is conflict-free.

13.43 Prove that a database scheme R is acyclic if and only if $*[\mathbf{R}]$ is equivalent to a conflict-free set of MVDs.

13.44* Let $\mathbf{R}$ be a database scheme. Recall that $[\cdot]_{*\mathbf{R}}$ is the window function defined by total projections of $*[\mathbf{R}]$-weak instances. Give an algorithm to compute $[X]_{*\mathbf{R}}$ that is polynomial in the size of the database.

13.45 Let $\mathbf{R}$ be a cyclic database scheme.

(a) Show that $\mathbf{R}$ can always be transformed to an acyclic scheme by the addition of a single relation scheme. (Don't think too hard.)

(b) Give an algorithm that is polynomial in the size of $\mathbf{R}$ that determines the size of the smallest relation scheme that will make $\mathbf{R}$ acyclic.

13.5 BIBLIOGRAPHY AND COMMENTS

The first manifestations of acyclic database schemes came from work on semijoins and on comparing pairwise consistency versus total consistency. The first definition of semijoin was given by Hall, Hitchcock, and Todd [1975], who called the operation "generalized intersection." There is mention of "semijoin" about the same time, but the operation referred to has nothing to do with what we are calling semijoin. Semijoins are used extensively in the distributed query processing algorithms for SDD-1, a distributed database system developed by Rothnie, Bernstein, *et al.* [1981]. Bernstein and Chiu [1981] were the first to connect join trees with full reducers,

although they handled only the case of semijoins on a single attribute. Bernstein and Goodman [1979a, 1979c] extended the theory to multiattribute semijoins. Theorem 13.1 is due to them. Several algorithms for finding minimum spanning trees may be found in Aho, Hopcroft, and Ullman [1974].

The interest in pairwise consistency and total consistency came from the problem of determining when a database is a projection of a common instance. Honeyman, Ladner, and Yannakakis [1980] showed the problem was NP-complete in general. Graham [1979] defined a large class of database schemes for which PC implies TC, but his class was a proper subset of the acyclic schemes. He gave the reduction algorithm, which was formulated independently by Yu and Ozsoyoglu [1979, 1980], although their algorithm is phrased in terms of join graphs. Honeyman [1980b] noted the connection between PC implying TC and the existence of full reducers, although his proof of equivalence is flawed.

Namibar [1979] was among the first researchers to formulate database scheme problems in terms of hypergraphs. The definition of acyclic database scheme, as well as the characterizations and equivalences not already attributed, comes from a series of papers by Fagin, Mendelzon, and Ullman [1980], Beeri, Fagin, Maier, Mendelzon, *et al.* [1981], and Beeri, Fagin, Maier, and Yannakakis [1981].

Bernstein and Goodman [1979b, 1980a] extend the theory of semijoins to involve inequality comparisons. Chiu and Ho [1980], and Chiu, Bernstein, and Ho [1980] give algorithms for finding the fastest full reducer for a given database state, provided a full reducer exists. Goodman and Shmueli [1980a, 1980b, 1981a, 1981b] examine a number of questions involving full-reducers and join trees, including reducers that use operations other than semijoins, the inapplicability of chase-type computation for determining if full reducers exist, generalizing cycles and cliques from graphs to hypergraphs and the complexity of modifying cyclic schemes to be acyclic. Chase [1981] also examines methods for eliminating acyclicity. Lien [1980] and Sciore [1981] look at sets of conflict-free MVDs, which can be used to characterize acyclic database schemes. Both argue that sets of MVDs that arise naturally from real world situations are conflict-free.

Yannakakis [1981] shows that acyclic schemes admit more efficient algorithms for some problems than cyclic schemes do. Katsuno [1981a] studies the interaction of acyclicity with FDs and MVDs. Maier and Ullman [1981] show that, in a certain sense, acyclic schemes are those where connections among sets of attributes are unique. Atzeni and Parker [1981] question the applicability of acyclic database schemes.

Exercises 13.7, 13.22, 13.41, 13.42, and 13.43 are from Beeri, Fagin, Maier, and Yannakakis [1981]. Exercise 13.13 is from Goodman and

Shmueli [1980a]. Exercise 13.28 is suggested by Bernstein and Chiu [1981]. Exercise 13.36 is answered by Fagin, Mendelzon, and Ullman [1980]. The "only if" direction of Exercise 13.37 is from Maier and Ullman [1981]. The "if" direction was noted by Kent Laver. Exercise 13.40 follows from Bernstein and Goodman [1979c]. The answer to Exercise 13.44 can be found in Yannakakis [1981]. Exercise 13.45b comes from Goodman and Shmueli [1981b].