APPENDIX C

Advanced Relational Database Design

In this appendix we cover advanced topics in relational database design. We first present the theory of multivalued dependencies, including a set of sound and complete inference rules for multivalued dependencies. We then present PJNF and DKNF, two normal forms based on classes of constraints that generalize multivalued dependencies.

C.1 Multivalued Dependencies

As we did for functional dependencies and 3NF and BCNF, we shall need to determine all the multivalued dependencies that are logically implied by a given set of multivalued dependencies.

C.1.1 Theory of Multivalued Dependencies

We take the same approach here that we did earlier for functional dependencies. Let D denote a set of functional and multivalued dependencies. The closure D^+ of D is the set of all functional and multivalued dependencies logically implied by D. As we did for functional dependencies, we can compute D^+ from D, using the formal definitions of functional dependencies and multivalued dependencies. However, it is usually easier to reason about sets of dependencies by using a system of inference rules.

The following list of inference rules for functional and multivalued dependencies is *sound* and *complete*. Recall that *sound* rules do not generate any dependencies that are not logically implied by D, and *complete* rules allow us to generate all dependencies in D^+. The first three rules are Armstrong's axioms, which we saw earlier in Chapter 7.

1. **Reflexivity rule**. If α is a set of attributes, and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.
2. **Augmentation rule**. If $\alpha \rightarrow \beta$ holds, and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.
3. **Transitivity rule**. If $\alpha \rightarrow \beta$ holds, and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.
4. **Complementation rule**. If $\alpha \twoheadrightarrow \beta$ holds, then $\alpha \twoheadrightarrow R - \beta - \alpha$ holds.
5. **Multivalued augmentation rule**. If $\alpha \twoheadrightarrow \beta$ holds, and $\gamma \subseteq R$ and $\delta \subseteq \gamma$, then $\gamma\alpha \twoheadrightarrow \delta\beta$ holds.
6. **Multivalued transitivity rule**. If $\alpha \twoheadrightarrow \beta$ holds, and $\beta \twoheadrightarrow \gamma$ holds, then $\alpha \twoheadrightarrow \gamma - \beta$ holds.
7. **Replication rule**. If $\alpha \rightarrow \beta$ holds, then $\alpha \twoheadrightarrow \beta$.
8. **Coalescence rule**. If $\alpha \twoheadrightarrow \beta$ holds, and $\gamma \subseteq \beta$, and there is a δ such that $\delta \subseteq R$, and $\delta \cap \beta = \emptyset$, and $\delta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ holds.

The bibliographical notes provide references to proofs that the preceding rules are sound and complete. The following examples provide insight into how the formal proofs proceed.

Let $R = (A, B, C, G, H, I)$ be a relation schema. Suppose that $A \twoheadrightarrow BC$ holds. The definition of multivalued dependencies implies that, if $t_1[A] = t_2[A]$, then there exist tuples t_3 and t_4 such that

$$
\begin{aligned}
&t_1[A] = t_2[A] = t_3[A] = t_4[A] \\
&t_3[BC] = t_1[BC] \\
&t_3[GHI] = t_2[GHI] \\
&t_4[GHI] = t_1[GHI] \\
&t_4[BC] = t_2[BC]
\end{aligned}
$$

The complementation rule states that, if $A \twoheadrightarrow BC$, then $A \twoheadrightarrow GHI$. Observe that t_3 and t_4 satisfy the definition of $A \twoheadrightarrow GHI$ if we simply change the subscripts.

We can provide similar justification for rules 5 and 6 (see Exercise C.2) using the definition of multivalued dependencies.

Rule 7, the replication rule, involves functional and multivalued dependencies. Suppose that $A \rightarrow BC$ holds on R. If $t_1[A] = t_2[A]$ and $t_1[BC] = t_2[BC]$, then t_1 and t_2 themselves serve as the tuples t_3 and t_4 required by the definition of the multivalued dependency $A \twoheadrightarrow BC$.

Rule 8, the coalescence rule, is the most difficult of the eight rules to verify (see Exercise C.4).

We can simplify the computation of the closure of D by using the following rules, which we can prove using rules 1 to 8 (see Exercise C.5):

- **Multivalued union rule**. If $\alpha \twoheadrightarrow \beta$ holds, and $\alpha \twoheadrightarrow \gamma$ holds, then $\alpha \twoheadrightarrow \beta\gamma$ holds.
- **Intersection rule**. If $\alpha \twoheadrightarrow \beta$ holds, and $\alpha \twoheadrightarrow \gamma$ holds, then $\alpha \twoheadrightarrow \beta \cap \gamma$ holds.

- **Difference rule**. If $\alpha \twoheadrightarrow \beta$ holds, and $\alpha \twoheadrightarrow \gamma$ holds, then $\alpha \twoheadrightarrow \beta - \gamma$ holds and $\alpha \twoheadrightarrow \gamma - \beta$ holds.

Let us apply our rules to the following example. Let $R = (A, B, C, G, H, I)$ with the following set of dependencies D given:

$$A \twoheadrightarrow B$$
$$B \twoheadrightarrow HI$$
$$CG \rightarrow H$$

We list several members of D^+ here:

- $A \twoheadrightarrow CGHI$: Since $A \twoheadrightarrow B$, the complementation rule (rule 4) implies that $A \twoheadrightarrow R - B - A$. $R - B - A = CGHI$, so $A \twoheadrightarrow CGHI$.
- $A \twoheadrightarrow HI$: Since $A \twoheadrightarrow B$ and $B \twoheadrightarrow HI$, the multivalued transitivity rule (rule 6) implies that$A \twoheadrightarrow HI - B$. Since $HI - B = HI$, $A \twoheadrightarrow HI$.
- $B \rightarrow H$: To show this fact, we need to apply the coalescence rule (rule 8). $B \twoheadrightarrow HI$ holds. Since $H \subseteq HI$ and $CG \rightarrow H$ and $CG \cap HI = \emptyset$, we satisfy the statement of the coalescence rule, with α being B, β being HI, δ being CG, and γ being H. We conclude that $B \rightarrow H$.
- $A \twoheadrightarrow CG$: We already know that $A \twoheadrightarrow CGHI$ and $A \twoheadrightarrow HI$. By the difference rule, $A \twoheadrightarrow CGHI - HI$. Since $CGHI - HI = CG$, $A \twoheadrightarrow CG$.

C.1.2 Dependency Preservation

The question of dependency preservation when we have multivalued dependencies is not as simple as it is when we have only functional dependencies.

A decomposition of schema R into schemas $R_1, R_2, \ldots, R_n$ is a **dependency-preserving decomposition** with respect to a set D of functional and multivalued dependencies if, for every set of relations $r_1(R_1), r_2(R_2), \ldots, r_n(R_n)$ such that for all i, r_i satisfies D_i (the restriction of D to R_i), there exists a relation $r(R)$ that satisfies D and for which $r_i = \Pi_{R_i}(r)$ for all i.

Let us apply the 4NF decomposition algorithm of Figure 7.17 to the schema $R = (A, B, C, G, H, I)$ with $D = \{A \twoheadrightarrow B, B \twoheadrightarrow HI, CG \rightarrow H\}$. We shall then test the resulting decomposition for dependency preservation. We first need to compute the closure of D. The nontrivial dependencies in closure include all the dependencies in D, and the multivalued dependency $A \twoheadrightarrow HI$, as we saw in Section C.1.1.

R is not in 4NF. Observe that $A \twoheadrightarrow B$ is not trivial, yet A is not a superkey. Using $A \twoheadrightarrow B$ in the first iteration of the **while** loop, we replace R with two schemas, (A, B) and (A, C, G, H, I). It is easy to see that (A, B) is in 4NF since all multivalued dependencies that hold on (A, B) are trivial. However, the schema (A, C, G, H, I) is not in 4NF. Applying the multivalued dependency $CG \twoheadrightarrow H$ (which follows from the given functional dependency $CG \rightarrow H$ by the replication rule), we replace (A, C, G, H, I) by the two schemas (C, G, H) and (A, C, G, I).

r_1:

A	B
a_1	b_1
a_2	b_1

r_2:

C	G	H
c_1	g_1	h_1
c_2	g_2	h_2

r_3:

A	I
a_1	i_1
a_2	i_2

r_4:

A	C	G
a_1	c_1	g_1
a_2	c_2	g_2

Figure C.1 Projection of relation r onto a 4NF decomposition of R.

A	B	C	G	H	I
a_1	b_1	c_1	g_1	h_1	i_1
a_2	b_1	c_2	g_2	h_2	i_2

Figure C.2 A relation $r(R)$ that does not satisfy $B \twoheadrightarrow HI$.

Schema (C, G, H) is in 4NF, but schema (A, C, G, I) is not. To see that (A, C, G, I) is not in 4NF, we note that since $A \twoheadrightarrow HI$ is in D^+, $A \twoheadrightarrow I$ is in the restriction of D to (A, C, G, I). Thus, in a third iteration of the **while** loop, we replace (A, C, G, I) by two schemas (A, I) and (A, C, G). The algorithm then terminates and the resulting 4NF decomposition is $\{(A, B), (C, G, H), (A, I), (A, C, G)\}$.

This 4NF decomposition is not dependency preserving, since it fails to preserve the multivalued dependency $B \twoheadrightarrow HI$. Consider Figure C.1, which shows the four relations that may result from the projection of a relation on (A, B, C, G, H, I) onto the four schemas of our decomposition. The restriction of D to (A, B) is $A \twoheadrightarrow B$ and some trivial dependencies. It is easy to see that r_1 satisfies $A \twoheadrightarrow B$, because there is no pair of tuples with the same A value. Observe that r_2 satisfies *all* functional and multivalued dependencies, since no two tuples in r_2 have the same value on any attribute. A similar statement can be made for r_3 and r_4. Therefore, the decomposed version of our database satisfies all the dependencies in the restriction of D. However, there is no relation r on (A, B, C, G, H, I) that satisfies D and decomposes into r_1, r_2, r_3, and r_4. Figure C.2 shows the relation $r = r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$. Relation r does not satisfy $B \twoheadrightarrow HI$. Any relation s containing r and satisfying $B \twoheadrightarrow HI$ must include the tuple $(a_2,\ b_1,\ c_2,\ g_2,\ h_1,\ i_1)$. However, $\Pi_{CGH}\ (s)$ includes a tuple $(c_2,\ g_2,\ h_1)$ that is not in r_2. Thus, our decomposition fails to detect a violation of $B \twoheadrightarrow HI$.

We have seen that, if we are given a set of multivalued and functional dependencies, it is advantageous to find a database design that meets the three criteria of

1. 4NF
2. Dependency preservation
3. Lossless join

If all we have are functional dependencies, then the first criterion is just BCNF.

We have seen also that it is not always possible to meet all three of these criteria. We succeeded in finding such a decomposition for the bank example, but failed for the example of schema $R = (A, B, C, G, H, I)$.

When we cannot achieve our three goals, we have to compromise on one of 4NF or dependency preservation.

C.2 Join Dependencies

We have seen that the lossless-join property is one of several properties of a good database design. Indeed, this property is essential: Without it, information is lost. When we restrict the set of legal relations to those satisfying a set of functional and multivalued dependencies, we are able to use these dependencies to show that certain decompositions are lossless-join decompositions.

Because of the importance of the concept of lossless join, it is useful to be able to constrain the set of legal relations over a schema *R* to those relations for which a given decomposition is a lossless-join decomposition. In this section, we define such a constraint, called a **join dependency**. Just as types of dependency led to other normal forms, join dependencies will lead to a normal form called **project-join normal form (PJNF)**.

C.2.1 Definition of Join Dependencies

Let *R* be a relation schema and $R_1, R_2, \ldots, R_n$ be a decomposition of *R*. The join dependency $*(R_1, R_2, \ldots, R_n)$ is used to restrict the set of legal relations to those for which $R_1, R_2, \ldots, R_n$ is a lossless-join decomposition of *R*. Formally, if $R = R_1 \cup R_2 \cup \ldots \cup R_n$, we say that a relation *r*(*R*) satisfies the *join dependency* $*(R_1, R_2, \ldots, R_n)$ if

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \cdots \bowtie \Pi_{R_n}(r)$$

A join dependency is *trivial* if one of the R_i is *R* itself.

Consider the join dependency $*(R_1, R_2)$ on schema *R*. This dependency requires that, for all legal $r(R)$,

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

Let *r* contain the two tuples t_1 and t_2, defined as follows:

$$\begin{array}{ll} t_1[R_1 - R_2] = (a_1, a_2, \ldots, a_i) & t_2[R_1 - R_2] = (b_1, b_2, \ldots, b_i) \\ t_1[R_1 \cap R_2] = (a_{i+1}, \ldots, a_j) & t_2[R_1 \cap R_2] = (a_{i+1}, \ldots, a_j) \\ t_1[R_2 - R_1] = (a_{j+1}, \ldots, a_n) & t_2[R_2 - R_1] = (b_{j+1}, \ldots, b_n) \end{array}$$

	$R_1 - R_2$	$R_1 \cap R_2$
$\Pi_{R_1}(t_1)$	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$
$\Pi_{R_1}(t_2)$	$b_1 \ldots b_i$	$a_{i+1} \ldots a_j$

	$R_1 \cap R_2$	$R_2 - R_1$
$\Pi_{R_2}(t_1)$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
$\Pi_{R_2}(t_2)$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$

Figure C.3 $\Pi_{R_1}(r)$ *and* $\Pi_{R_2}(r)$.

Thus, $t_1[R_1 \cap R_2] = t_2[R_1 \cap R_2]$, but t_1 and t_2 have different values on all other attributes. Let us compute $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$. Figure C.3 shows $\Pi_{R_1}(r)$ and $\Pi_{R_2}(r)$. When we compute the join, we get two tuples in addition to t_1 and t_2, shown by t_3 and t_4 in Figure C.4.

If $*(R_1, R_2)$ holds, then, whenever we have tuples t_1 and t_2, we must also have t_3 and t_4. Thus, Figure C.4 shows a tabular representation of the join dependency $*(R_1, R_2)$. Compare Figure C.4 with Figure 7.14, in which we gave a tabular representation of $\alpha \twoheadrightarrow \beta$. If we let $\alpha = R_1 \cap R_2$ and $\beta = R_1$, then we can see that the two tabular representations in these figures are the same. Indeed, $*(R_1, R_2)$ is just another way of stating $R_1 \cap R_2 \twoheadrightarrow R_1$. Using the complementation and augmentation rules for multivalued dependencies, we can show that $R_1 \cap R_2 \twoheadrightarrow R_1$ implies $R_1 \cap R_2 \twoheadrightarrow R_2$. Thus, $*(R_1, R_2)$ is equivalent to $R_1 \cap R_2 \twoheadrightarrow R_2$. This observation is not surprising in light of the fact we noted earlier that R_1 and R_2 form a lossless-join decomposition of R if and only if $R_1 \cap R_2 \twoheadrightarrow R_2$ or $R_1 \cap R_2 \twoheadrightarrow R_1$.

Every join dependency of the form $*(R_1, R_2)$ is therefore equivalent to a multivalued dependency. However, there are join dependencies that are not equivalent to any multivalued dependency. The simplest example of such a dependency is on schema $R = (A, B, C)$. The join dependency

$$*((A, B), (B, C), (A, C))$$

is not equivalent to any collection of multivalued dependencies. Figure C.5 shows a tabular representation of this join dependency. To see that no set of multivalued dependencies logically implies $*((A, B), (B, C), (A, C))$, we consider Figure C.5 as a relation $r(A, B, C)$, as in Figure C.6. Relation r satisfies the join dependency

	$R_1 - R_2$	$R_1 \cap R_2$	$R_2 - R_1$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$b_1 \ldots b_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$b_1 \ldots b_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$

Figure C.4 Tabular representation of $*(R_1, R_2)$.

A	B	C
a_1	b_1	c_2
a_2	b_1	c_1
a_1	b_2	c_1
a_1	b_1	c_1

Figure C.5 Tabular representation of $*((A, B), (B, C), (A, C))$.

$*((A, B), (B, C), (A, C))$, as we can verify by computing

$$\Pi_{AB}(r) \bowtie \Pi_{BC}(r) \bowtie \Pi_{AC}(r)$$

and by showing that the result is exactly r. However, r does not satisfy any nontrivial multivalued dependency. To see that it does not, we verify that r fails to satisfy any of $A \twoheadrightarrow B$, $A \twoheadrightarrow C$, $B \twoheadrightarrow A$, $B \twoheadrightarrow C$, $C \twoheadrightarrow A$, or $C \twoheadrightarrow B$.

Just as a multivalued dependency is a way of stating the independence of a pair of relationships, a join dependency is a way of stating that the members of a *set* of relationships are all independent. This notion of independence of relationships is a natural consequence of the way that we generally define a relation. Consider

Loan_info_schema = (*branch_name, customer_name, loan_number, amount*)

from our banking example. We can define a relation *loan_info* (*Loan_info_schema*) as the set of all tuples on *Loan_info_schema* such that

- The loan represented by *loan_number* is made by the branch named *branch_name*.
- The loan represented by *loan_number* is made to the customer named *customer_name*.
- The loan represented by *loan_number* is in the amount given by *amount*.

The preceding definition of the *loan_info* relation is a conjunction of three predicates: one on *loan_number* and *branch_name*, one on *loan_number* and *customer_name*, and one on *loan_number* and *amount*. Surprisingly, it can be shown that the preceding intuitive definition of *loan_info* logically implies the join dependency *((*loan_number, branch_name*), (*loan_number, customer_name*), (*loan_number, amount*)).

A	B	C
a_1	b_1	c_2
a_2	b_1	c_1
a_1	b_2	c_1
a_1	b_1	c_1

Figure C.6 Relation $r(A, B, C)$.

Thus, join dependencies have an intuitive appeal and correspond to one of our three criteria for a good database design.

For functional and multivalued dependencies, we were able to give a system of inference rules that are sound and complete. Unfortunately, no such set of rules is known for join dependencies. It appears that we must consider more general classes of dependencies than join dependencies to construct a sound and complete set of inference rules. The bibliographical notes contain references to research in this area.

C.2.2 Project-Join Normal Form

Project-join normal form (*PJNF*) is defined in the same way as BCNF and 4NF, except that join dependencies are used. A relation schema R is in PJNF with respect to a set D of functional, multivalued, and join dependencies if, for all join dependencies in D^+ of the form $*(R_1, R_2, \ldots, R_n)$, where each $R_i \subseteq R$ and $R = R_1 \cup R_2 \cup \ldots \cup R_n$, at least one of the following holds:

- $*(R_1, R_2, \ldots, R_n)$ is a trivial join dependency.
- Every R_i is a superkey for R.

A database design is in PJNF if each member of the set of relation schemas that constitutes the design is in PJNF. PJNF is called *fifth normal form* (*5NF*) in some of the literature on database normalization.

Consider again our banking example. Given the join dependency *((*loan_number, branch_name*), (*loan_number, customer_name*), (*loan_number, amount*)), *Loan_info_schema* is not in PJNF. To put *Loan_info_schema* into PJNF, we must decompose it into the three schemas specified by the join dependency: (*loan_number, branch_name*), (*loan_number, customer_name*), and (*loan_number, amount*).

Because every multivalued dependency is also a join dependency, it is easy to see that every PJNF schema is also in 4NF. Thus, in general, we may not be able to find a dependency-preserving decomposition into PJNF for a given schema.

C.3 Domain-Key Normal Form

The approach we have taken to normalization is to define a form of constraint (functional, multivalued, or join dependency), and then to use that form of constraint to define a normal form. *Domain-key normal form* (*DKNF*) is based on three notions.

1. **Domain declaration**. Let A be an attribute, and let **dom** be a set of values. The domain declaration $A \subseteq$ **dom** requires that the A value of all tuples be values in **dom**.
2. **Key declaration**. Let R be a relation schema with $K \subseteq R$. The key declaration **key** (K) requires that K be a superkey for schema R—that is, $K \rightarrow R$. Note that all key declarations are functional dependencies but not all functional dependencies are key declarations.

3. **General constraint**. A *general constraint* is a predicate on the set of all relations on a given schema. The dependencies that we have studied in this chapter are examples of general constraints. In general, a general constraint is a predicate expressed in some agreed-on form, such as first-order logic.

We now give an example of a general constraint that is not a functional, multivalued, or join dependency. Suppose that all accounts whose *account_number* begins with the digit 9 are special high-interest accounts with a minimum balance of $2500. Then, we include as a general constraint, "If the first digit of *t*[*account_number*] is 9, then *t*[*balance*] $\geq$ 2500."

Domain declarations and key declarations are easy to test in a practical database system. General constraints, however, may be extremely costly (in time and space) to test. The purpose of a DKNF database design is to allow us to test the general constraints using only domain and key constraints.

Formally, let **D** be a set of domain constraints and let **K** be a set of key constraints for a relation schema *R*. Let **G** denote the general constraints for *R*. Schema *R* is in DKNF if **D** $\cup$ **K** logically imply **G**.

Let us return to the general constraint that we gave on accounts. The constraint implies that our database design is not in DKNF. To create a DKNF design, we need two schemas in place of *Account_schema*:

Regular_acct_schema = (*account_number, branch_name, balance*)
Special_acct_schema = (*account_number, branch_name, balance*)

We retain all the dependencies that we had on *Account_schema* as general constraints. The domain constraints for *Special_acct_schema* require that, for each account,

- The account number begins with 9.
- The balance is greater than 2500.

The domain constraints for *Regular_acct_schema* require that the account number does not begin with 9. The resulting design is in DKNF, although the proof of this fact is beyond the scope of this text.

Let us compare DKNF to the other normal forms that we have studied. Under the other normal forms, we did not take into consideration domain constraints. We assumed (implicitly) that the domain of each attribute was some infinite domain, such as the set of all integers or the set of all character strings. We allowed key constraints (indeed, we allowed functional dependencies). For each normal form, we allowed a restricted form of general constraint (a set of functional, multivalued, or join dependencies). Thus, we can rewrite the definitions of PJNF, 4NF, BCNF, and 3NF in a manner that shows them to be special cases of DKNF.

We now present a DKNF-inspired rephrasing of our definition of PJNF. Let $R = (A_1, A_2, \ldots, A_n)$ be a relation schema. Let $\text{dom}(A_i)$ denote the domain of attribute A_i, and let all these domains be infinite. Then all domain constraints **D** are of the form $A_i \subseteq \text{dom}(A_i)$. Let the general constraints be a set **G** of functional, multivalued, or join dependencies. If *F* is the set of functional dependencies in **G**, let the set **K** of key

constraints be those nontrivial functional dependencies in F^+ of the form $\alpha \rightarrow R$. Schema R is in PJNF if and only if it is in DKNF with respect to **D**, **K**, and **G**.

A consequence of DKNF is that all insertion and deletion anomalies are eliminated.

DKNF represents an "ultimate" normal form because it allows arbitrary constraints, rather than dependencies, yet it allows efficient testing of these constraints. Of course, if a schema is not in DKNF, we may be able to achieve DKNF via decomposition, but such decompositions, as we have seen, are not always dependency-preserving decompositions. Thus, although DKNF is a goal of a database designer, it may have to be sacrificed in a practical design.

C.4 Summary

In this chapter we presented the theory of multivalued dependencies, including a set of sound and complete inference rules for multivalued dependencies.

We then presented two more normal forms based on more general classes of constraints. Join dependencies are a generalization of multivalued dependencies, and lead to the definition of PJNF. DKNF is an idealized normal form that may be difficult to achieve in practice. Yet DKNF has desirable properties that should be included to the extent possible in a good database design.

Exercises

C.1 List all the nontrivial multivalued dependencies satisfied by the relation in Figure C.7.

C.2 Use the definition of multivalued dependency (Section 7.6.1) to argue that each of the following axioms is sound:

- **a.** The complementation rule
- **b.** The multivalued augmentation rule
- **c.** The multivalued transitivity rule

C.3 Use the definitions of functional and multivalued dependencies (Sections 7.4 and 7.6.1) to show the soundness of the replication rule.

C.4 Show that the coalescence rule is sound. (*Hint*: Apply the definition of $\alpha \twoheadrightarrow \beta$ to a pair of tuples t_1 and t_2 such that $t_1[\alpha] = t_2[\alpha]$. Observe that since $\delta \cap \beta = \emptyset$, if two tuples have the same value on $R - \beta$, then they have the same value on δ.)

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

Figure C.7 Relation of Exercise C.1.

C.5 Use the axioms for functional and multivalued dependencies to show that each of the following rules is sound:

a. The multivalued union rule
b. The intersection rule
c. The difference rule

C.6 Let $R = (A, B, C, D, E)$, and let M be the following set of multivalued dependencies

$$A \twoheadrightarrow BC$$
$$B \twoheadrightarrow CD$$
$$E \twoheadrightarrow AD$$

List the nontrivial dependencies in M^+.

C.7 Give a lossless-join decomposition of schema R in Exercise C.6 into 4NF.

C.8 Give an example of relation schema R and a set of dependencies such that R is in 4NF, but is not in PJNF.

C.9 Explain why PJNF is a normal form more desirable than is 4NF.

C.10 Rewrite the definitions of 4NF and BCNF using the notions of domain constraints and general constraints.

C.11 Explain why DKNF is a highly desirable normal form, yet is one that is difficult to achieve in practice.

Bibliographical Notes

The notions of 4NF, PJNF, and DKNF are from Fagin [1977], Fagin [1979], and Fagin [1981], respectively. The synthesis approach to database design is discussed in Bernstein [1976].

Join dependencies were introduced by Rissanen [1979]. Sciore [1982] gives a set of axioms for a class of dependencies that properly includes the join dependencies. In addition to their use in PJNF, join dependencies are central to the definition of universal relation databases. Fagin et al. [1982] introduces the relationship between join dependencies and the definition of a relation as a conjunction of predicates (see Section C.2.1). This use of join dependencies has led to a large amount of research into *acyclic* database schemas. Intuitively, a schema is acyclic if every pair of attributes is related in a unique way. Formal treatment of acyclic schemas appears in Fagin [1983] and in Beeri et al. [1983].

Additional dependencies are discussed in detail in Maier [1983]. Inclusion dependencies are discussed by Casanova et al. [1984] and Cosmadakis et al. [1990]. Template dependencies are covered by Sadri and Ullman [1982]. Mutual dependencies are examined by Furtado [1978] and by Mendelzon and Maier [1979].