Chapter 12

NULL VALUES, PARTIAL INFORMATION AND DATABASE SEMANTICS

The reader is warned that the topics in this chapter are matters of personal taste. Whether the definitions and approaches offered seem right depends on individual intuition. This chapter presents some of the problems that arise from the assumption that all information fits into the relational model. The problems are not completely solved, but some partial solutions are offered as guidance for future work. The following line, overheard at a discussion of data semantics, gives the proper frame of mind: "It all makes sense if you squint a little and don't think too hard."

Underlying the relational database model is a blithe assumption that the information to be represented fits nicely into little boxes arranged in rectangular tables. The assumption can fail for two reasons:

1. The structure of the information does not fit the mold.
2. The information fits the mold, but part of the information is missing.

As an example of the first, recall the problem in the *pinfo*(PART# SUBPART PARTNAME) relation of Section 10.2. We needed an artificial value for SUBPARTOF whenever a part was not a subpart. We chose 0 as the artificial value. There is some danger in that approach. The domain calculus expression

$$\{x(\text{PART\#})\, y(\text{PART\#2}) \mid \exists z_1\, \exists z_2\, \exists z_3\, (\mathit{pinfo}(x\ z_1\ z_2) \land \mathit{pinfo}(y\ z_1\ z_3))\}$$

might at first seem the correct query to find all pairs of parts that are subparts of the same part. However, it also associates all pairs of parts that are not subparts of any part. The problem is that we need a way to represent *partial information*—we would like to have tuples that range over only a part of the relation scheme.

As an aside, we note that there are some cases where "no value" could properly be considered a value. Consider a relation *books*(TITLE FIRSTNAME MIDDLENAME LASTNAME) for recording titles and authors of books. If some author has no middle name, a special domain value could properly be used to represent the middle name. If two full names agree in the first name and the last name, and have no middle name, they are the same name. In this example, it is appropriate to augment the domain for MIDDLENAME with a "none" or "does not exist" element that is treated identically to any other domain value. Note the difference from the SUBPARTOF example. Contrast the statements "Two authors with no middle names can be considered to have the same middle name" and "Two parts that are not subparts can be considered to be subparts of a common part."

Turning to the second reason given above, recall the relation *usedon*(PART# PTYPE NUSED), also from Section 10.2. Suppose we know that part 318 is used on a 1011, but we do not know the number used. We would like to store a tuple with the information that we do have; a tuple t with t(PART#) = 318 and t(PTYPE) = 1011. What value should we store for t(NUSED)? Unlike the SUBPARTOF example, we know there is some domain value that could correctly fill the slot. However, using any particular domain value would give incorrect answers for queries, unless we happened to guess the correct value. We need a means to represent *unknown values*.

We shall first look at the problem of unknown values. We introduce *nulls* to represent the unknown values. We then show that functional dependencies and *marked nulls* may be used to fill in unknown values at times. We consider *existence constraints* as a means to control the use of nulls. Finally, we introduce *possibility functions* as a tool for comparing extensions of relational algebra to handle nulls. We then turn to the partial information problem. This problem is related to the problem of treating a database as a single semantic entity. We introduce *window functions* as a tool for viewing a database as a unit.

12.1 NULLS

The term *null* has been applied widely in database literature to special domain values that arise for a variety of reasons. Here we use it narrowly to mean "value exists but unknown." We use the symbol $\perp$ to denote a null. Although we may use the null symbol several places in a relation, each occurrence represents a potentially different unknown value.

Example 12.1 Suppose the personnel department at our airline is rather snoopy, and maintains a relation *history*(EMPLOYEE SALARY PREVEMP

PREVJOB PREVSAL). The relation records the previous employer, job title, and salary of an employee, as well as the current salary. However, personnel may not be able to wheedle all this information from every employee, so the relation may be incomplete, as shown in Table 12.1.

Table 12.1 Relation Table Comprising Employee *history*.

history(	EMPLOYEE	SALARY	PREVEMP	PREVJOB	PREVSAL)
t_1	Lambert	39,500	SWA	pilot	36,000
t_2	Larson	24,100	$\perp$	$\perp$	$\perp$
t_3	Lathen	17,300	WIA	clerk	$\perp$
t_4	Liu	18,260	WIA	$\perp$	17,800

Since the problems of unknown values and partial information are so slippery, and have not been completely solved, our treatment will tend toward examples more than proofs. However, what we lack in rigor we shall make up in notation.

Notation A tuple containing 0 or more nulls is *partial*. A tuple with no nulls is *total*. Thus, every total tuple is a partial tuple (similarly to partial and total functions). A tuple t whose scheme includes attribute A is *definite* on A, written $t(A)\downarrow$, if $t(A)$ is not null. This notation extends to sets of attributes: $t(X)\downarrow$ means $t(A)\downarrow$ for every attribute $A \in X$. We use $t\downarrow$ to mean that t is total. If t is a tuple over scheme R, we let $DEF(t) = \{A \in R | t(A)\downarrow\}$. For tuples t and u on the same scheme, t *subsumes* u, written $t \geq u$, if $u(A)\downarrow$ implies $u(A) = t(A)$ for every attribute A in X. If $t \geq u$ and $t\downarrow$, we call t an *extension* of u, written $t \downarrow\geq u$.

Example 12.2 In Table 12.1, all the tuples are partial, while t_1 is also total. For t_2, t_2(EMPLOYEE)$\downarrow$, t_2(SALARY)$\downarrow$ and $DEF(t_2)$ = EMPLOYEE SALARY. The tuple $t = \langle$Larson 24,100 $\perp$ manager $\perp\rangle$ subsumes tuple t_2. For the tuple $u = \langle$Larson 24,100 WIA manager 22,050$\rangle$, $u \downarrow\geq t$.

More Notation A relation r is *total*, written $r\downarrow$, if all its tuples are total. Relations containing 0 or more nulls are *partial*. For relation scheme R, we let $Rel\uparrow(R)$ be the set of all partial relations over R and let $Rel(R)$ be the set of all total relations over R. Through Section 12.4, relation will mean partial relation. For relations r and s over R, r *subsumes* s, written $r \geq s$, if for every tuple $t_s \in s$ there is a tuple $t_r \in r$ such that $t_r \geq t_s$. If $r \geq s$ and $s \geq r$, we write $r \simeq s$. If r is total, then r is an *extension* of s, written $r \downarrow\geq s$.

If r can be obtained from s by changing some nulls in s to values, then r *augments* s, written $r \succeq s$. Clearly, $r \succeq s$ implies $r \geq s$. If r is total, then r

completes s, or r is a *completion* of s, written $r \downarrow\succeq s$. If $r \succeq s$ and $s \succeq r$, then $r = s$ (see Exercise 12.4). Evidently, r augments s if there is a mapping α of the tuples of s *onto* the tuples of r such that $\alpha(t) \geq t$ for every tuple $t \in s$. Note that the term *completion* is given a different meaning in this chapter than in Section 8.7.

Example 12.3 The relation *history*$'$ in Table 12.2 is an extension of the relation *history* in Table 12.1, but not a completion, because of the tuples ⟨Lathen 17,300 WIA clerk 16,400⟩ and ⟨Lathen 17,300 WIA clerk 16,850⟩. The relation *history*$''$ in Table 12.3 is a completion of *history*. Note that in general, neither $r \geq s$ nor $r \succeq s$ imply $|r| \geq |s|$.

Table 12.2 Extension of Relation *history*.

history$'$(EMPLOYEE	SALARY	PREVEMP	PREVJOB	PREVSAL)
Lambert	39,500	SWA	pilot	36,000
Larson	24,000	WIA	manager	22,050
Lathen	17,300	WIA	clerk	16,400
Lathen	17,300	WIA	clerk	16,850
Liu	18,260	WIA	agent	17,800

Table 12.3 Completion of Relation *history*.

history$''$(EMPLOYEE	SALARY	PREVEMP	PREVJOB	PREVSAL)
Lambert	39,500	SWA	pilot	36,000
Larson	24,000	WIA	manager	22,050
Lathen	17,300	WIA	clerk	16,400
Liu	18,260	WIA	agent	17,800

We can view a partial relation as a set of axioms about the total relation that the partial relation represents.

Example 12.4 We can view the relation *history* in Table 12.1 as a set of axioms (in domain calculus notation) about a total relation *history*:

1. *history*(Lambert 39,500 SWA pilot 36,000)
2. $\exists x_1 \exists x_2 \exists x_3$ *history*(Larson 24,100 x_1 x_2 x_3)
3. $\exists x_4$ *history*(Lathen 17,300 WIA clerk x_4)
4. $\exists x_5$ *history*(Liu 18,260 WIA x_5 17,800).

If r is a partial relation, then any extension of r can be interpreted as a finite model satisfying the axioms denoted by r. Also, if $r \geq s$, then, as sets of axioms about a total relation, r logically implies s.

For the moment, we shall be interested in those extensions that are completions. That is, every tuple in a partial relation is seen as representing a single tuple of a total relation. However, two tuples in the partial relation may represent the same tuple in the total relation. The principle is that we do not assume any more information than we need to in order to satisfy the axioms denoted by the partial relation. There are some subtle problems here. If relation r' is obtained from relation r by removing a tuple t from r, where t is subsumed by another tuple in r, then $r' \geq r$ and so $r' \simeq r$. Thus r' and r have the same set of extensions. While $r' \succeq r$ (why?), it is *not* necessarily true that $r \succeq r'$. In fact, it is never true that $r \succeq r'$, since that would imply $r = r'$, by Exercise 12.4. Thus there are completions of r that are not completions of r'.

The relationship $\succeq$ does not correspond to logical implication of sets of axioms, because we use the form of the axioms to limit the possible models. Logicians may be troubled by a system where models are restricted by the syntax of the axioms; where seemingly equivalent sets of axioms have different models. Linguists will be less troubled, for they know "It's not what you say, it's how you say it." Researchers in artificial intelligence won't see what all the fuss is about. We shall see in Chapter 14 that even the assumption that our models are finite has interesting effects upon the implication of dependencies. One way around the problem of subsumed tuples is to consider only relations with no such tuples, although we do not adopt that assumption here.

Given that we allow partial relations in a database, the problem of evaluating queries immediately arises. We briefly describe the *null substitution principle* for interpreting calculus formulas over partial relations. We examine only the single-relation case: multiple-relation formulas can be treated similarly. We define an interpretation function $\hat{I}$ that maps calculus formulas to the set {*true*, *unknown*, *false*}. Let f be a tuple calculus formula that mentions only a single relation, $r(R)$. Assume further that f has no free variables. Let $I_s(f)$ stand for the usual interpretation of formula f, using total relation $s(R)$ in place of r. We define

$\hat{I}(f) =$ *true* if $I_s(f) =$ *true* for every completion s of r,
$\hat{I}(f) =$ *false* if $I_s(f) =$ *false* for every completion s of r, and
$\hat{I}(f) =$ *unknown* otherwise.

That is, $\hat{I}(f)$ is *true* exactly when every possible substitution of values for nulls in r makes f a true formula in the usual sense.

Example 12.5 Let *history* be the relation in Table 12.1, and let R denote its scheme. Consider the formula f_1 defined as

$$\exists x(\mathrm{R}) \in \textit{history}\ (x(\mathrm{PREVEMP}) = \text{“WIA”} \wedge x(\mathrm{PREVJOB}) = \text{“clerk”}).$$

$\hat{I}(f_1) = \textit{true}$, since no matter how the null in tuple t_3 of *history* is filled in, $t_3(\mathrm{PREVEMP}) = \text{“WIA”} \wedge t_3(\mathrm{PREVJOB}) = \text{“clerk”}$ will be true. Next, consider the formula f_2 defined as

$$\exists x(R) \in \textit{history}\ (x(\mathrm{EMPLOYEE}) = \text{“Liu”} \wedge \forall y(R) \in \textit{history}\ (x(\mathrm{PREVSAL}) \geq y(\mathrm{PREVSAL}))).$$

$\hat{I}(f_2) = \textit{false}$, because no matter how nulls are filled in, t_4 is the only tuple that could make $x(\mathrm{EMPLOYEE}) = \text{“Liu”}$ true and choosing t_1 for y makes the formula as a whole false. Finally, consider the formula f_3 defined as

$$\exists x(R) \in \textit{history}\ (x(\mathrm{EMPLOYEE}) = \text{“Lambert”} \wedge \forall y(R) \in \textit{history}\ (x(\mathrm{PREVSAL}) \geq y(\mathrm{PREVSAL}))).$$

$\hat{I}(f_3) = \textit{unknown}$, since f_3 can become either true or false depending on how $t_2(\mathrm{PREVSAL})$ and $t_3(\mathrm{PREVSAL})$ are filled in.

A problem with the null substitution principle is that it does not admit a recursive definition of $\hat{I}$. That is, we cannot define $\hat{I}(f)$ in terms of $\hat{I}$'s value on the immediate subformulas of f. The next example shows that if $f = g \vee h$, then $\hat{I}(f)$ cannot be defined as $\hat{I}(g) \vee \hat{I}(h)$.

Example 12.6 Let *history* be the relation in Table 12.1 and let R represent its scheme. Let f be

$$\exists x(R) \in \textit{history}\ (x(\mathrm{PREVEMP}) = \text{“WIA”} \wedge x(\mathrm{PREVSAL}) \leq 16{,}000) \vee \exists y(R) \in \textit{history}\ (y(\mathrm{PREVJOB}) = \text{“clerk”} \wedge y(\mathrm{PREVSAL}) > 16{,}000).$$

$\hat{I}(f) = \textit{true}$, but $\hat{I}(g) = \hat{I}(h) = \textit{unknown}$, where g and h are the two disjuncts of f.

While there is no recursive definition of $\hat{I}$, it is not necessary to try all possible substitutions of domain values for nulls in a formula f to compute $\hat{I}(f)$.

We need only look at enough domain values to allow any atom to become either true or false (if possible). For the formula f in Example 12.6, 16,000 and 15,999 are the only values that need be considered for nulls in the PREVSAL column of *history*. Nevertheless, evaluating $\hat{I}(f)$ is computationally hard, since it is basically the problem of testing for a tautology.

12.2 FUNCTIONAL DEPENDENCIES AND NULLS

We may have some restrictions on the total relation that a partial relation can represent. In particular, it may be that the total relation must satisfy some FDs.

Example 12.7 Consider the version of *history* shown in Table 12.4. We assume that the total relation that *history* represents has EMPLOYEE as a key. Hence, we would allow the completion *history'* shown in Table 12.5, but not the completion *history"* shown in Table 12.6.

Table 12.4 Relation *history* with EMPLOYEE as a Key.

history(EMPLOYEE	SALARY	PREVEMP	PREVJOB	PREVSAL)
Lathen	17,300	WIA	clerk	$\perp$
$\perp$	18,260	WIA	$\perp$	17,800

Table 12.5 Completion of Relation *history* (allowed).

history'(EMPLOYEE	SALARY	PREVEMP	PREVJOB	PREVSAL)
Lathen	17,300	WIA	clerk	16,400
Liu	18,260	WIA	agent	17,800

Table 12.6 Completion of Relation *history* (not allowed).

history"(EMPLOYEE	SALARY	PREVEMP	PREVJOB	PREVSAL)
Lathen	17,300	WIA	clerk	16,400
Lathen	18,260	WIA	agent	17,800

One thing we would like to test given a relation $r \in Rel^{\uparrow}(R)$ and a set of FDs F is whether any completion of r satisfies F.

Definition 12.1 Let $r \in Rel^{\uparrow}(R)$ and let F be a set of FDs over R. Relation r is *permissible* with respect to F if some completion s of r satisfies F. A completion that satisfies F is a *permissible completion* under F.

Example 12.8 Let F be {EMPLOYEE → SALARY PREVEMP PREVJOB PREVSAL}. The version of *history* shown in Table 12.4 is permissible with respect to F, while the version of Table 12.7 is not.

Table 12.7 Relation *history* (not permissible with respect to F).

history(EMPLOYEE	SALARY	PREVEMP	PREVJOB	PREVSAL)
Lathen	17,300	WIA	clerk	⊥
Lathen	⊥	SWA	⊥	17,800

We seek a test for permissibility of partial relations. As it turns out, FDs are not just constraints on possible completions. They can also be used to fill in nulls. For a particular null, there may be only a single way to fill in that null in order to satisfy a set of FDs.

Example 12.9 Consider the relation *vacation*(EMPLOYEE YEARS ANNUALDAYS ACCDAYS) in Table 12.8 that gives the number of years employees have been with the airline, how many days of vacation they accumulate annually, and the total days accumulated. We assume the set of FDs F = {EMPLOYEE → YEARS ANNUALDAYS ACCDAYS, YEARS → ANNUALDAYS} applies. In any completion of vacation that satisfies F, t_1(ACCDAYS) = 17, t_2(YEARS) = 3, t_2(ANNUALDAYS) = 21 and t_3(ANNUALDAYS) = 21. Thus we can fill in these nulls to get a more complete relation, *vacation* ′, as shown in Table 12.9. It may seem that all the problems go away if *vacation* is decomposed using the FD YEARS → ANNUALDAYS, but there are similar examples where all the FDs involved have keys on the left. In later sections we shall see cases where we want to combine several relations in order to fill in nulls.

Table 12.8 The Relation *vacation*.

vacation(EMPLOYEE		YEARS	ANNUALDAYS	ACCDAYS)
t_1	Udall	3	21	⊥
t_2	Udall	⊥	⊥	17
t_3	Unthank	3	⊥	⊥

Table 12.9 Partial completion of the Relation *vacation*.

vacation ′(EMPLOYEE		YEARS	ANNUALDAYS	ACCDAYS)
$t_1 = t_2$	Udall	3	21	17
t_3	Unthank	3	21	⊥

Our strategy for testing permissibility will be to fill in as many nulls as possible using information given by the FDs. How may we fill in nulls in a relation r given a set of FDs F? The last example gives one obvious rule as to how a null may be filled in. Let $X \rightarrow A$ be an FD in F. Let t_1 and t_2 be two tuples in r such that $t_1(X)\downarrow$, $t_2(X)\downarrow$, and $t_1(X) = t_2(X)$. Suppose that $t_1(A) = \perp$, but that $t_2(A) = a$. We can change $t_1(A)$ to a, since in any permissible completion of r, $t_1(A)$ must be a to satisfy $X \rightarrow A$.

The rule just given is correct, but it does not go far enough.

Example 12.10 Consider the relation $r(A\ B\ C\ D)$ in Figure 12.1, and the set of FDs $F = \{A \rightarrow C, C \rightarrow D\}$. A moment's inspection will show that $t_1(D) = d$ in any permissible completion of r, although the rule just given cannot be used to fill in any nulls in r.

$r($	A	B	C	$D)$
t_1	a	b	$\perp$	$\perp$
t_2	a	$\perp$	$\perp$	d

Figure 12.1

In the last example, $t_1(D)$ can be filled in using the FDs in F^+, namely $A \rightarrow D$. There are examples where even F^+ does not suffice (see Exercise 12.10). The problem is that we need to indicate that two nulls represent the same value, even though that value is unknown. In the last example, we need to indicate $t_1(C) = t_2(C)$ in any permissible completion of r. To that end, we introduce *marked nulls*. We subscript the null symbol to get an infinite collection of different nulls: $\{\perp_1, \perp_2, \perp_3, \ldots\}$. Marked nulls will be assumed distinct unless they have the same subscript. Let t_1 and t_2 be two tuples over scheme R involving marked nulls. If A is an attribute in R, t_1 and t_2 *agree on* A if either

1. $t_1(A)\downarrow$ and $t_2(A)\downarrow$ and $t_1(A) = t_2(A)$, or
2. $t_1(A) = \perp_i$, $t_2(A) = \perp_j$ and $i = j$.

The tuples agree on a set of attributes X in R if they agree on every attribute in X. For the rest of this section, all nulls in partial relations are assumed to be marked. Any relation with unmarked nulls may be converted to one with marked nulls by appending distinct subscripts to unmarked nulls. We consider two relations in $Rel\uparrow(R)$ to be the same if they are identical except for a one-to-one renaming of marked nulls.

Example 12.11 The relation r in Figure 12.1 could appear as shown in Figure 12.2 with marked nulls.

$r($	A	B	C	$D)$
t_1	a	b	$\perp_1$	$\perp_2$
t_2	a	$\perp_3$	$\perp_4$	d

Figure 12.2

We need two definitions before giving the rule for filling in marked nulls.

Definition 12.2 Let $r \in Rel\uparrow(R)$, let $X \rightarrow A$ be an FD over R and let t_1 and t_2 be tuples in r that agree on X. If t_1 and t_2 are both definite on A, but $t_1(A) \neq t_2(A)$, r has a *hard violation* of $X \rightarrow A$. If t_1 and t_2 disagree on A, and at least one is null, then r has a *soft violation* of $X \rightarrow A$. Hard violations cannot be removed by filling in nulls, while soft violations can.

The *fill-in rule for marked nulls* mimics the F-rule of the chase computation. The rule fills in or equates marked nulls whenever they participate in a soft violation. Let $r \in Rel\uparrow(R)$ and let F be a set of FDs over R. Let t_1 and t_2 be tuples in r that participate in a soft violation of the FD $X \rightarrow A$ in F.

1. If $t_1(A)\downarrow$ and $t_2(A) = \perp_i$, change every occurrence of $\perp_i$ in r to $t_1(A)$.
2. If $t_2(A)\downarrow$ and $t_1(A) = \perp_i$, change every occurrence of $\perp_i$ in r to $t_2(A)$.
3. If $t_1(A) = \perp_i$, $t_2(A) = \perp_j$, and $i < j$, change every occurrence of $\perp_j$ to $\perp_i$; if $i > j$, change every occurrence of $\perp_i$ to $\perp_j$.

These changes remove the soft violation, although they may introduce new violations.

Example 12.12 Starting with relation r in Figure 12.2 and the FDs $F = \{A \rightarrow C, C \rightarrow D\}$, we apply the fill-in rule using $A \rightarrow C$ to change $\perp_4$ to $\perp_1$. We may then use the fill-in rule with $C \rightarrow D$ to change $\perp_2$ to d. The result is shown in Figure 12.3.

$r($	A	B	C	$D)$
t_1	a	b	$\perp_1$	d
t_2	a	$\perp_3$	$\perp_1$	d

Figure 12.3

The repeated application of the fill-in rule for a set of FDs F is similar to the chase computation on a tableau. Values correspond to distinguished variables and marked nulls correspond to nondistinguished variables. However, where a tableau has only one distinguished symbol per column, a relation can have many values in a column. The result of repeated application of the fill-in rule need not be unique.

Example 12.13 Let $r(A\ B\ C)$ be the relation in Figure 12.4. Let $F = \{A \rightarrow B,\ C \rightarrow B\}$. We see there are two ways to fill in $\perp_2$, one using $A \rightarrow B$ and the other using $C \rightarrow B$.

$r(A$	B	$C\ )$
1	2	$\perp_1$
1	$\perp_2$	3
$\perp_3$	4	3

Figure 12.4

While the relation in the example above could be filled in two ways using the given set of FDs, either way gives a hard violation. We shall show, and it should be apparent, that any relation with a hard violation has no permissible completions. To assure a unique result when applying the fill-in rule to a relation, we assume when a relation has a hard violation, the fill-in rule changes it to some special value, which we call HV. We let $nchase_F(r)$ stand for the result of applying the fill-in rule with FDs from f to relation r until no changes can be made. $Nchase_F(r)$ will either be a relation or the special value HV.

We want to show that r is permissible with respect to F exactly when $nchase_F(r)$ is not HV. $Nchase_F(r)$ can have multiple copies of a marked null. We update the definitions of *augment* and *completion* to handle the multiple copies. We still view a relation with marked nulls as a set of axioms (actually a single axiom) about a total relation that the partial relation represents. The difference is that tuples no longer generate separately quantified axioms.

Example 12.14 The relation $r(A\ B\ C)$ in Figure 12.5 represents the axiom (in domain calculus form):

$$\exists x_1\ \exists x_2\ (r(1\ 2\ x_1) \wedge r(x_2\ 2\ 3) \wedge r(x_2\ 4\ x_1)).$$

Note this formula is not equivalent to

$$\exists x_1\ (r(1\ 2\ x_1)) \wedge \exists x_2\ (r(x_2\ 2\ 3)) \wedge \exists x_1\ \exists x_2\ (r(x_2\ 4\ x_1)).$$

r(A	B	C)
1	2	$\perp_1$
$\perp_2$	2	3
$\perp_2$	4	$\perp_1$

Figure 12.5

We leave an updated definition of subsumes to Exercise 12.12. For r and s in $Rel\uparrow(R)$, r *augments* s, written $r \succeq s$, if r is obtained from s by filling in some nulls in s with values or other nulls. The restriction is that whenever a null $\perp_i$ is changed, every copy of $\perp_i$ must be changed in the same way. Also, we do not allow a marked null to appear in more than one column of a relation. If, in addition, $r\downarrow$, we say r is a *completion* of s, written $r \downarrow > s$. These definitions reduce to the ones for the unmarked null case when neither r nor s contains a repeated marked null.

We now show two results. Assume HV has no permissible completions under any set of FDs. First, $nchase_F(r)$ and r always have the same permissible completions under F. Second, if $nchase_F(r) \neq HV$, then r has at least one permissible completion.

We actually show that if r' is obtained from r by a single application of the fill-in rule using an FD form F, then r and r' have the same permissible completions under F. If r' is actually HV, then r had a hard violation. The tuples in r constituting the hard violation will still exhibit the hard violation when filled in. Therefore, r has no permissible completions. If r' is a relation, then r' came from r by replacing a marked null with another marked null or with a value. In either case, $r' \succeq r$, so any permissible completion of r' is a permissible completion of r.

Consider how r' was obtained from r. There must be an FD $X \rightarrow A$ in F and tuples t_1 and t_2 in r such that t_1 and t_2 agree on X. Further, either $t_1(A) = \perp_i$ and $t_2(A) = a$, or $t_1(A) = a$ and $t_2(A) = \perp_i$, or $t_1(A) = \perp_i$ and $t_2(A) = \perp_j$, where $i \neq j$. In any completion s of r, $t_1(X) = t_2(X)$. For s to satisfy F, t_1 and t_2 must agree on A. In the first two cases, both must be a on A, as is the case in r'. If s is a permissible completion of r, then s is a permissible completion of r'. Similarly, in the third case, $\perp_i$ and $\perp_j$ must be replaced by the same value if s is to satisfy F. Hence, again, if s is permissible for r, it is a completion for r', and so a permissible completion for r'. We conclude r and r' have the same permissible completions under F.

From this argument it follows that r and $nchase_F(r)$ have the same permissible completions under F.

Consider the case where $r^* = nchase_F(r)$ is not HV. Form a completion s of r^* by replacing every marked null in r^* with a value distinct from any value in r^* and distinct from the value used to replace any other null. We claim s satisfies F (see Exercise 12.14).

Example 12.15 Let $r(A\ B\ C\ D\ E)$ be the relation in Figure 12.6. Let $F = \{A \to B, B\ D \to C\}$. $Nchase_F(r)$ is the relation r^* shown in Figure 12.7. Filling in nulls with distinct values gives the completion s shown in Figure 12.8, which indeed satisfies F.

$r(A$	B	C	D	$E\ )$
1	$\perp_1$	$\perp_2$	2	3
1	$\perp_3$	4	5	$\perp_4$
6	7	8	5	9
1	$\perp_5$	$\perp_6$	2	9

Figure 12.6

$r^*(A$	B	C	D	$E\ )$
1	$\perp_1$	$\perp_2$	2	3
1	$\perp_1$	4	5	$\perp_4$
6	7	8	5	9
1	$\perp_1$	$\perp_2$	2	9

Figure 12.7

$s(A$	B	C	D	$E\ )$
1	10	11	2	3
1	10	4	5	12
6	7	8	5	9
1	10	11	2	9

Figure 12.8

To see why s must satisfy F, choose an arbitrary FD $X \to A$ from F. Let t_1 and t_2 be tuples in s that agree on X. In r^*, t_1 and t_2 must agree on X, otherwise filling in nulls would have made them disagree. Since r^* cannot have a soft or hard violation, $t_1(A) = t_2(A)$ in r^*. Hence $t_1(A) = t_2(A)$ in s. Thus s satisfies $X \to A$. Since $X \to A$ was arbitrary, s satisfies F and is therefore a permissible completion of r^* and r.

Nchase gives a method for testing whether a relation r is permissible with respect to a set of FDs F, as well as filling in all nulls possible, assuming infinite domains (see Exercise 12.15). There is still some information about the permissible completions of r that $nchase_F(r)$ does not capture: certain nulls may not take on certain values. If some domains are finite, rather than infinite as we have assumed, it may be possible to determine values for more nulls using FDS.

Example 12.16 Consider the relation *restrooms* in Table 12.10. Assume it obeys the FD FLOOR SEX → ROOM# (only one restroom of each type per floor). We can conclude that t_3(SEX) $\neq$ *women*. If the domain of sex is {*men*, *women*}, then we can change t_3(SEX) to *men*, since it must be so in any permissible completion of restrooms.

Table 12.10. The Relation *restrooms*.

restrooms(	FLOOR	ROOM#	SEX)
t_1	1	8B	women
t_2	2	8B	women
t_3	2	8D	$\perp_1$

There does not seem to be any labeling scheme to capture information about which values a null may not assume and which nulls must be filled in with different values in a permissible completion. There are other problems to be examined in connection with nulls. Can marked nulls be used to fill in values across relations in a database? Can other dependencies be used to fill in values in partial relations? Should relations such as

$r(A$	$B\,)$
1	$\perp_1$
1	$\perp_2$

be allowed? Note that r has more completions than

$s(A$	$B\,)$
1	$\perp_1$.

In Section 12.4 we shall examine some ways other than completions to denote what a partial relation represents.

12.3 CONSTRAINTS ON NULLS

In practice, there will usually be restrictions on where nulls should appear in a relation. Typically, nulls are forbidden in any components of the primary key of a relation. We give a means to express such restrictions. For this section it does not matter if nulls are marked or not.

Definition 12.3 Let R be a relation scheme. An *existence constraint* (EC) over R is a statement of the form $X \urcorner Y$ (read "X requires Y") for X and Y subsets of R. A relation $r \in Rel\uparrow(R)$ satisfies $X \urcorner Y$ if for every tuple t in r, $t(X)\downarrow$ implies $t(Y)\downarrow$.

Example 12.17 Let $r(A\ B\ C\ D)$ be the relation shown in Figure 12.9. Relation r satisfies the ECs $D \urcorner A$, $C\ D \urcorner A$ and $\emptyset \urcorner B$. The last EC requires that all tuples be definite on B. The EC A $\urcorner D$ is not satisfied (because of t_2), nor is $B\ A \urcorner C$.

r	$(A$	B	C	$D)$
t_1	1	2	$\perp$	3
t_2	1	4	$\perp$	$\perp$
t_3	$\perp$	2	5	$\perp$
t_4	6	2	7	8

Figure 12.9

A relation satisfies an EC if each of its tuples individually satisfies the EC. This situation is unlike FDs, where pairs of tuples must be considered. Suprisingly, inference axioms for ECs take the exact form of inference axioms for FDs. For example, if $X \urcorner Y$, then $X\ A \urcorner Y$ for any attribute A. Also, if $X \urcorner Y$ and $Y \urcorner Z$, we may conclude $X \urcorner Z$ (see Exercise 12.16). If we have only ECs to check as constraints, it is a simple matter to test whether a given update will cause a relation to violate the ECs. Any deletion is acceptable. Insertion and modification depend only on the tuple involved. When FDs also are being enforced, and are used to fill in nulls, other tuples may be affected by an insertion or modification. The tuple to be inserted may initially satisfy the ECs, but some nulls may be filled in using FDs and cause an EC violation. It is also possible that a tuple already in the relation will be made to violate the ECs.

Example 12.18 The relation $r(A\ B\ C\ D)$ in Figure 12.10 satisfies the EC A $\urcorner B\ C$, and is permissible with respect to the FD $B \rightarrow A$. Adding the tuple $t_3 = \langle 5\ 1\ 4\ \perp \rangle$ will cause t_1 to violate $A \urcorner B\ C$ when $t_1(A)$ is filled in as 5 using $B \rightarrow A$.

There remains much work to be done on which sets of FDs and ECs behave nicely together (see Exercise 12.17).

r	A	B	C	D
t_1	$\perp$	1	$\perp$	6
t_2	2	3	4	$\perp$

Figure 12.10

12.4 RELATIONAL ALGEBRA AND PARTIAL RELATIONS

We now turn to extending relational algebra to relations with nulls. For this section, we let $Rel\uparrow$ and Rel denote the sets of all partial and total relations whose schemes are taken from some fixed universe $\mathbf{U}$ of attributes. Also, we return to the use of unmarked nulls. We shall view a relation r in $Rel\uparrow(R)$ as denoting a set of total relations from $Rel(R)$. This set of possibilities we call $POSS(r)$. In previous sections we have used $POSS(r) = \{s \mid s \text{ is a completion of } r\}$. Here we shall consider other definitions for $POSS$. Relative to $POSS$, we consider ways to extend relational operators from Rel to $Rel\uparrow$. Suppose we want to extend the join operator to partial relations. Ideally we would like an extended operator $\bowtie'$ such that

$$POSS(r \bowtie' s) = POSS(r) \bowtie POSS(s)$$

for r and s in $Rel\uparrow$. By $P_1 \bowtie P_2$, for sets of relations P_1 and P_2, we mean

$$\{q_1 \bowtie q_2 \mid q_1 \in P_1 \quad \text{and} \quad q_2 \in P_2\}.$$

We shall first discuss what properties $POSS$ should possess, and what constitutes a reasonable extension of a relational operator relative to a given possibility function. We then look at proposals for an extended join operator and discuss their shortcomings. Finally, we consider for which definitions of $POSS$ there exist reasonable extensions of the relational operators.

12.4.1 Possibility Functions

Definition 12.4 A *possibility function* is a function $POSS$: $Rel\uparrow \rightarrow 2^{Rel}$. That is, $POSS$ assigns every partial relation a set of total relations. We require that if $r \in Rel\uparrow(R)$, then $POSS(r) \subseteq Rel(R)$.

As the definition stands, there is nothing to indicate that a possibility function is supposed to represent possible total relations for a partial relation. The definition allows a possibility function that maps every partial relation to the empty relation over the same scheme. While such a stonewalling possibility function might have use in political circles, we choose to impose additional restrictions on possibility functions.

Definition 12.5 Let q and r be relations over scheme R such that $q \downarrow\geq r$. If for every tuple $t_q \in q$ there is a tuple $t_r \in r$ such that $t_q \geq t_r$, we say q is a *close extension* of r. If no proper subrelation of q is an extension of r, but q is an extension of r, then q is a *minimal extension* of r. Any minimal extension is also a close extension (see Exercise 12.18).

Example 12.19 Consider the relation $r(A\ B\ C)$ in Figure 12.11. Relation q in Figure 12.12 is a close extension of r, but not a minimal extension, because tuples t_2 and t_3 could be eliminated. Relation q' in Figure 12.13 is a minimal extension of r.

$r(A$	B	$C)$
1	2	$\perp$
$\perp$	2	3
1	$\perp$	3

Figure 12.11

q	$(A$	B	$C)$
t_1	1	2	3
t_2	4	2	3
t_3	1	5	3

Figure 12.12

$q'(A$	B	$C)$
1	2	6
7	2	3
1	8	3

Figure 12.13

Any possibility function *POSS* gives rise to a partial ordering on $Rel^{\dagger}(R)$ for any R. If $r, s \in Rel^{\dagger}(R)$, r is *stronger than* s, relative to *POSS*, written $r \sqsupseteq_{POSS} s$, if $POSS(r) \subseteq POSS(s)$. If $r \sqsupseteq_{POSS} s$ and $s \sqsupseteq_{POSS} r$, we say r and s are *equally strong*, written $r \equiv_{POSS} s$. We write $\sqsupseteq$ and $\equiv$ for $\sqsupseteq_{POSS}$ and $\equiv_{POSS}$ when *POSS* is understood.

We can now describe further restrictions on possibility functions.

Definition 12.6 A possibility function *POSS* is *reasonable* if

1. every element of $POSS(r)$ is an extension of r,
2. $POSS(r)$ contains every minimal extension of r, and
3. for every r and s over the same scheme, $s \in POSS(r)$ if and only if $s \sqsupseteq r$ and $s\downarrow$.

Definition 12.7 *POSS* is *closed* if $POSS(r)$ contains only close extensions of r.

Condition 1 arises from viewing r as axioms about a total relation. The condition says that every relation in $POSS(r)$ satisfies the axioms that r denotes. Condition 2 says there are no "hidden axioms." Any total relation that satisfies the axioms r denotes, and has no superfluous tuples in that regard, is in $POSS(r)$. Condition 3 captures the idea that a total relation interpreted as a set of axioms agrees with the relation as a model for a set of axioms. If *POSS* is closed, $s\downarrow$ implies $POSS(s) = \{s\}$, so condition 3 is trivially satisfied.

A closed possibility function corresponds to a "closed world assumption" in heuristic inference systems. The closed world assumption states that what is not provably true is assumed to be false. In our situation, where some of our information is incomplete, it is not clear which statements should be construed as true. This uncertainty allows several variations on the closed world assumption.

Example 12.20 We have already seen one possibility function, $POSS_C(r) = \{s | s \downarrow \succeq r\}$. The subscript C stands for "completion." We leave it to the reader to show that $POSS_C$ is reasonable and closed. The possibility function $POSS_O(r) = \{s | s \downarrow \geq r\}$ is also reasonable, but it is not closed. (The O is for "open.") It is the only possibility function we shall consider that is not closed. Notice $r \sqsupseteq_{POSS_O} s$ holds exactly when $r \geq s$ (see Exercise 12.25). The possibility function $POSS_E(r) = \{s | s \downarrow \geq r$ and s has an even number of tuples$\}$ is not reasonable, since it violates conditions 2 and 3.

Henceforth, we consider only reasonable possibility functions.

If $r \in Rel(R)$, $s \in Rel\uparrow(R)$ and $r \sqsupseteq s$ for some possibility function *POSS*, then $r \in POSS(s)$ and $r \geq s$, by conditions 3 and 1 of the definition of reasonable possibility function. These results also hold when $r \in Rel\uparrow(R)$.

Lemma 12.1 Let *POSS* be a possibility function, $\sqsupseteq$ its associated strength ordering, and r and s partial relations on scheme R. Then $r \sqsupseteq s$ implies $r \geq s$.

Proof Suppose $r \not\geq s$. Let t_s be a tuple in s that is not subsumed by any tuple in r. It is possible to construct an extension q of r such that for any tuple $t_q \in q$, $t_q \not\geq t_s$. Furthermore, we can assume q is minimal. (Otherwise, remove tuples from q.) Since q is a minimal extension of r, $q \in POSS(r)$, but $q \notin POSS(s)$, since $q \not\geq s$. Therefore, $POSS(r) \not\subseteq$ POSS(s) and so $r \not\sqsupseteq s$. We have shown the contrapositive of the lemma.

Corollary If $r \equiv s$, then $r \simeq s$.

We shall see cases where the converse of Lemma 12.1 fails: $r \geq s$ does not imply $r \sqsupseteq s$. Let $r \lessgtr s$ mean $r \not\geq s$ and $s \not\geq r_m$ (r and s are incomparable under subsumption). We may conclude from the lemma that $r \lessgtr s$ implies $r \not\equiv s$.

12.4.2 Generalizing the Relational Operators

We shall use possibility functions to characterize when the generalization of an algebraic operator from *Rel* to *Rel*↑ is reasonable. There is one criterion, however, that is independent of the choice of possibility function. We want the generalized operator to agree with the regular operator on *Rel*.

Definition 12.8 Let γ be an operator on *Rel* and let γ' be an operator on *Rel*↑; γ' is *faithful* to γ if

1. when γ and γ' are unary operators, $\gamma(r) = \gamma'(r)$ for every $r \in Rel$ for which $\gamma(r)$ is defined, or
2. when γ and γ' are binary operators, $r\,\gamma\,s = r\,\gamma'\,s$ for every $r, s \in Rel$ for which $r\,\gamma\,s$ is defined.

The next definition gives the ideal behavior of a generalized operator.

Definition 12.9 Let γ be an operator on *Rel* and let γ' be an operator on *Rel*↑. Operator γ' is a *precise* generalization of γ relative to possibility function *POSS* if

1. when γ and γ' are unary operators, $POSS(\gamma'(r)) = \gamma(POSS(r))$ for every $r \in Rel$↑; or
2. when γ and γ' are binary operators, $POSS(r\ \gamma'\ s) = POSS(r)\ \gamma\ POSS(s)$ for every $r, s \in$ Rel↑.

Here, for sets P_1 and P_2 of total relations,

$$\gamma(P_1) = \{\gamma(q) | q \in P_1\} \text{ and}$$
$$P_1 \,\gamma\, P_2 = \{q_1 \,\gamma\, q_2 | q_1 \in P_1, q_2 \in P_2\}.$$

Unfortunately, for some possibility functions, $\gamma(POSS(r))$ or $POSS(r)\ \gamma\ POSS(s)$ may not be regular enough to describe as $POSS(q)$ for any q. In such cases we settle for a generalization of γ that captures everything in $\gamma(POSS(r))$ or $POSS(r)\ \gamma\ POSS(s)$ and as little extra as possible.

Definition 12.10 Let γ be an operator on *Rel* and let γ' be an operator on *Rel*↑. Let *POSS* be a possibility function. Operator γ' is *adequate* for γ relative to *POSS* if

1. when γ and γ' are unary operators, $POSS(\gamma'(r)) \supseteq \gamma(POSS(r))$ for every $r \in$ *Rel*↑, or
2. when γ and γ' are binary operators, $POSS(r\ \gamma'\ s) \supseteq POSS(r)\ \gamma\ POSS(s)$ for every $r, s \in$ *Rel*↑.

Operator γ' is *restricted* for γ relative to *POSS* if

1. when γ and γ' are unary operators, for every $r \in$ *Rel*↑, there is no q in *Rel*↑ such that
   $POSS(\gamma'(r)) \supsetneq POSS(q) \supseteq \gamma(POSS(r))$, or
2. When γ and γ' are binary operators, for every $r, s \in$ *Rel*↑, there is no q in *Rel*↑ such that
   $POSS(r\ \gamma'\ s) \supsetneq POSS(q) \supseteq POSS(r)\ \gamma\ POSS(s)$.

Clearly, if γ' is precise for γ, then γ' is adequate and restricted for γ. We shall content ourselves with an adequate and restricted generalization when no precise generalization is available. We would also like the generalized operators to have properties that the regular operator possesses, such as commutativity or associativity. For example, if γ is an associative binary operator, we want a generalization γ' to satisfy

$$(q\ \gamma'\ r)\ \gamma'\ s \equiv q\ \gamma'\ (r\ \gamma'\ s)$$

for $q, r, s \in$ *Rel*↑. We now consider some generalizations of equijoin and natural join that have been proposed previously.

Example 12.21 Codd defined a "maybe" equijoin for use with relations containing nulls. We examine a simple form of it. Let $r(R)$ and $s(S)$ be relations in *Rel*↑, with $A \in R$, $C \in S$, and $R \cap S = \emptyset$. We extend the equijoin $[A = C]$ as $[A = C]'$ where

$$r[A = C]'\,s = \{t(RS) \mid t(R) \in r, t(S) \in s \text{ and } t(A) = t(C), \text{ or at least one of } t(A) \text{ and } t(C) \text{ is null}\}.$$

(This definition is actually a combination of Codd's "definite" and "maybe" equijoins.) If we let $r(A\ B)$ and $s(C\ D)$ be as shown in Figure 12.14, then $r[A = C]'\,s$ is given in Figure 12.15. Note that this generalization of equijoin cannot be used to derive a generalization of natural join because the A and C values of a tuple do not necessarily agree.

$r(A$	$B)$
1	2
4	$\perp$

$s(C$	$D)$
2	3
$\perp$	5

Figure 12.14

$r[A = C]'s(A$	B	C	$D)$
1	2	2	3
4	$\perp$	2	3
1	2	$\perp$	5
4	$\perp$	$\perp$	5

Figure 12.15

If we take the three relations

$q(A$	$B)$
1	2

$r(C$	D)
$\perp$	3

$s(E$	$F)$
4	5

and compute $(q\ [A = C]'\ r)\ [C = E]'\ s$, we get

$q'(A$	B	C	D	E	$F)$
1	2	$\perp$	3	4	5

Note that for no (reasonable) possibility function *POSS* is the empty relation in *POSS*(q'). However, if $[A = C]'$ and $[C = E]'$ are adequate for $[A = C]$ and $[C = E]$ under *POSS*, then

$$POSS(q') \supseteq POSS(q[A = C]'\,r)\ [C = E]\ POSS(s) \supseteq (POSS(q)\ [A = C]\ POSS(r))\ [C = E]\ POSS(s),$$

which contains the empty relation. Take any close extensions $\hat{q}$, $\hat{r}$, and $\hat{s}$ for q, r, and s;

$$(\hat{q}\ [A = C]\ \hat{r})\ [C = E]\ \hat{s}$$

is the empty relation. We see that this generalization of equijoin is not adequate for any choice of *POSS*. The problem is that the null in relation r is assumed to equal 1 in one case and 4 in another. We shall see in the next section a modification of Codd's notions that does work out.

Example 12.22 LaCroix and Pirotte propose a generalized natural join operator, denoted $\overset{+}{\bowtie}$, that guarantees every tuple enters a join. Tuples that do not join with other tuples are padded with nulls and added to the result. For the relations $r(A\ B)$ and $s(B\ C)$ in Figure 12.16, $r \overset{+}{\bowtie} s$ is shown in Figure 12.17. LaCroix and Pirotte give several choices as to how to proceed when a null appears in a join column. No matter what choice is taken, $\overset{+}{\bowtie}$ is not associative. Consider the relations

$q(A$	$B)$
1	2

$r(B$	$C)$
2	3

$s(A$	$C)$
1	4

Computing $(q \overset{+}{\bowtie} r) \overset{+}{\bowtie} s$ we get

$q'(A$	B	$C)$
1	2	3
1	$\perp$	4

while $q \overset{+}{\bowtie} (r \overset{+}{\bowtie} s)$ gives

$q''(A$	B	$C)$
1	2	4
$\perp$	2	3

Now $q' \not\asymp q''$, so for any reasonable possibility function $q' \not\equiv q''$, by Lemma 12.1.

$r(A$	$B)$
1	2
1	4

$s(B$	$C)$
2	3
5	6

Figure 12.16

$r \overset{+}{\bowtie} s$(*A*	*B*	*C*)
1	2	3
1	4	⊥
⊥	5	6

Figure 12.17

Example 12.23 Zaniolo proposed a generalization of join, which we shall denote $\bowtie^{Z}$, where tuples join if on each common attribute either they agree or exactly one is null. If one tuple is null on an attribute, the joined tuple takes its value from the other tuple. If a tuple does not join with any tuples, it is padded with nulls and added to the result. For the relations $r(A\ B)$ and $s(B\ C)$ in Figure 12.18, $r \bowtie^{Z} s$ is shown in Figure 12.19.

r(*A*	*B*)
1	2
4	5

s(*B*	*C*)
2	3
⊥	6
7	8

Figure 12.18

$r \bowtie^{Z} s$(*A*	*B*	*C*)
1	2	3
1	2	6
4	5	6
⊥	7	8

Figure 12.19

There is no reasonable possibility function for which $\bowtie^{Z}$ is associative. Consider

q(*A*	*B*)
1	2
⊥	5

r(*A*	*B*)
⊥	3
⊥	6

s(*A*	*B*)
⊥	4
7	8

Computing $(q \bowtie^{Z} r) \bowtie^{Z} s$ gives

q′(*A*	*B*	*C*	*D*)
1	2	3	4
1	2	6	4
7	5	⊥	8

while $q \bowtie^Z (r \bowtie^Z s)$ gives

$q''(A$	B	C	$D)$
7	5	3	8
7	5	6	8
1	2	$\perp$	4

Since $q' \not\geq q''$, we conclude from Lemma 12.1 that $q' \not\equiv q''$ for any reasonable possibility function.

12.4.3 Specific Possibility Functions

In this section we examine several possibility functions to see if adequate and restricted generalized operators exist for them. We shall limit our attention to four operators: (natural) join, union, project, and select on equality.

We first consider $POSS_O$, the "open" possibility function. Recall

$$POSS_O(r) = \{s \mid s \downarrow \geq r\},$$

and that for $POSS_O$, $r \sqsupseteq s$ exactly when $r \geq s$.

Consider a join operator for $POSS_O$; call it $\bowtie^O$. It cannot be precise. For partial relations $r(R)$ and $s(S)$, $POSS_O(r) \bowtie POSS_O(s)$ is a subset of $SAT(*[R\ S])$. For any relation $q \in Rel\uparrow(R\ S)$, $POSS_O(q)$ is not a subset of $SAT(*[R\ S])$ (see Exercise 12.26). We can give an adequate and restricted definition for $\bowtie^O$. Let $r(R)$ and $s(S)$ be in $Rel\uparrow$, with $R \cap S = X$. Let

$$r \bowtie^O s = \{t(R\ S) \mid \text{there are } t_r \in r \text{ and } t_s \in s \text{ with } t_r(X)\downarrow, \\ t_s(X)\downarrow,\ t(R) = t_r, \text{ and } t(S) = t_s\}.$$

We join tuples from r and s if they are definite on and agree on X.

Example 12.24 The generalized join of relations $r(A\ B\ C)$ and $s(B\ C\ D)$ in Figure 12.20 is shown in Figure 12.21.

$r(A$	B	$C)$
1	2	$\perp$
$\perp$	3	4
1	3	5

$s(B$	C	$D)$
2	7	3
$\perp$	4	8
3	4	9
3	4	10
3	5	$\perp$

Figure 12.20

$r \bowtie^O s(A$	B	C	$D)$
$\perp$	3	4	9
$\perp$	3	4	10
1	3	5	$\perp$

Figure 12.21

As defined, $\bowtie^O$ is adequate. Let $q = r \bowtie^O s$ and let $\hat{q}$ be any relation in $POSS_O(r) \bowtie POSS_O(s)$. We must show $\hat{q} \geq q$, from which it immediately follows that $\hat{q} \downarrow\geq q$, hence $\hat{q} \in POSS_O(q)$, hence $POSS_O(q) \supseteq POSS_O(r) \bowtie POSS_O(s)$. Let $\hat{q} = \hat{r} \bowtie \hat{s}$ for $\hat{r} \in POSS_O(r)$ and $\hat{s} \in POSS_O(s)$. Let t_q be any tuple in q. There must exist tuples $t_r \in r$ and $t_s \in s$ such that $t_r(X)\downarrow$, $t_s(X)\downarrow$, $t_q(R) = t_r$ and $t_q(S) = t_s$. We conclude $t_r(X) = t_s(X)$. Since $\hat{r} \geq r$, there must be a tuple $t_{\hat{r}} \in \hat{r}$ such that $t_{\hat{r}} \geq t_r$ and so $t_{\hat{r}}(X) = t_r(X)$. Likewise, there is a tuple $t_{\hat{s}} \in \hat{s}$ such that $t_{\hat{s}} \geq t_s$ and so $t_{\hat{s}}(X) = t_s(X)$. Relation $\hat{q}$ must contain a tuple $t_{\hat{q}}$ such that $t_{\hat{q}}(R) = t_{\hat{r}}$ and $t_{\hat{q}}(S) = t_{\hat{s}}$. It follows $t_{\hat{q}} \geq t_q$. Since the choice of t_q was arbitrary, $\hat{q} \geq q$, as desired. We leave the proof that $\bowtie^O$ is restricted as Exercise 12.27.

Union has a precise generalization for $POSS_O$. Let r and s be in $Rel\uparrow(R)$, and let

$$r \cup^O s = \{t \mid t \in r \text{ or } t \in s\}.$$

Suppose $q = r \cup^O s$. We first show $POSS_O(q) \supseteq POSS_O(r) \cup POSS_O(s)$. (Union here is an element-wise union.) Let $\hat{q} \in POSS_O(r) \cup POSS_O(s)$. There must be $\hat{r} \in POSS_O(r)$ and $\hat{s} \in POSS_O(s)$ such that $\hat{q} = \hat{r} \cup \hat{s}$. Let t_q be a tuple in q. Either $t_q \in r$ or $t_q \in s$. If $t_{\hat{q}} \in \hat{r}$, there is a tuple $t_{\hat{q}} \in \hat{r}$, and hence in $\hat{q}$, such that $t_{\hat{q}} \geq t_q$. If $t_q \in s$, there is similarly a tuple $t_{\hat{q}}$ in $\hat{q}$ with $t_{\hat{q}} \geq t_q$. We conclude $\hat{q} \geq q$, hence $\hat{q} \downarrow\geq q$ and so $\hat{q} \in POSS_O(q)$.

Suppose now that $\hat{q} \in POSS_O(q)$. Since $q \geq r$ (Why?), $\hat{q} \downarrow\geq r$ and so $\hat{q} \in POSS_O(r)$. Similarly, $\hat{q} \in POSS_O(s)$. Therefore $\hat{q} \in POSS_O(r) \cup POSS_O(s)$ and so $POSS_O(q) \subseteq POSS_O(r) \cup^O POSS_O(s)$. We conclude $\cup^O$ is precise for $POSS_O$.

Project also has a precise generalization for $POSS_O$. We leave the definition to the reader (see Exercise 12.28).

We turn to select with equality comparisons between two attributes or an attribute and a value. Let $r \in Rel\uparrow(R)$ and let $A \in R$. We define

$$\sigma^O_{A=a}(r) = \{t(R) \mid t \in r \text{ and } t(A) = a\}.$$

Example 12.25 Let $r(A\ B\ C)$ be the relation in Figure 12.22; $\sigma^O_{A=1}(r)$ is shown in Figure 12.23

$r(A$	B	$C)$
1	1	2
1	$\perp$	3
$\perp$	4	5
4	4	6

Figure 12.22

$\sigma^O_{A=1}(r)(A$	B	$C)$
1	1	2
1	$\perp$	3

Figure 12.23

For comparing two attributes $A, B \in R$ we have

$$\sigma^O_{A=B}(r) = \{t(R) \mid t \in r,\ t(A)\downarrow,\ t(B)\downarrow \text{ and } t(A) = t(B)\}.$$

Example 12.26 The relation r in Figure 12.22, $\sigma^O_{A=B}(r)$ is shown in Figure 12.24.

$\sigma^O_{A=B}(r)(A$	B	$C)$
1	1	2
4	4	6

Figure 12.24

These two definitions are not precise, but precise definitions are not possible. For $\sigma^O_{A=a}$, note that for any relation $s \in \sigma_{A=a}(POSS_O(r))$, every tuple $t \in s$ has $t(A) = a$. For any relation q, $POSS_O(q)$ contains relations whose tuples are not all a on A. The two definitions are adequate and restricted (see Exercise 12.29).

We now examine closed possibility functions. There does not seem to be any closed possibility function that has adequate and restricted generalizations for all of join, union, project, and select. While we do not exhaust all reasonable closed possibility functions, we do treat the three most natural ones.

The most liberal definition for a closed possibility function is

$$POSS_{CE}(r) = \{s \mid s \text{ is a close extension of } r\}.$$

For $POSS_{CE}$, $r \geq s$ does *not* imply $r \sqsupseteq s$. Consider relations

$r(A$	$B)$
1	2
$\perp$	2

and

$s(A$	$B)$
1	2

We have $r \geq s$, but

$\hat{r}(A$	$B)$
1	2
3	2

is a close extension of r but not of s. Hence, $POSS_{CE}(r) \not\subseteq POSS_{CE}(s)$ and so $r \not\sqsupseteq s$. $POSS_{CE}$ has adequate and restricted generalizations of union and project (see Exercise 12.31a). No adequate generalization exists for select. Consider $\sigma_{A=1}(POSS_{CE}(r))$ for

$r(A$	$B)$
$\perp$	2 .

There is no relation q in $Rel\!\uparrow\!(A\ B)$ such that $POSS_{CE}(q) \supseteq \sigma_{A=1}(POSS_{CE}(r))$. $POSS_{CE}(q)$ can contain either non-empty relations, or just the empty relation, while $\sigma_{A=1}(POSS_{CE}(r))$ contains the empty relation and the relation

$\hat{r}(A$	$B)$
1	2 .

A similar argument shows there is no adequate generalization of join for $POSS_{CE}$ (see Exercise 12.31b).

Another closed possibility function is

$$POSS_C(r) = \{s \mid s \downarrow\geq r\}.$$

There is no adequate generalization of project for $POSS_C$. Consider

$r(A$	$B)$
$\perp$	1
$\perp$	2 .

The set π_A $(POSS_C(r))$ is all one- and two-tuple total relations on scheme A. There is no relation $q \in Rel\mathfrak{t}(A)$ such that $POSS_C(q)$ contains all such relations. The same problem arises for the possibility function $POSS_{ME}$ where

$$POSS_{ME}(r) = \{s \mid s \text{ is a minimal extension of } r\}.$$

$POSS_{ME}$ is the most conservative closed possibility function.

The chances seem bleak for a closed possibility function with adequate and restricted definitions for join, union, select, and project. Biskup has a possibility function using a modification of partial relations for which such generalizations exist. His approach divides a relation into *sure* tuples and *maybe* tuples. We shall call this variety of partial relation a *partitioned relation*. A partitioned relation can be viewed as an ordered pair of partial relations. If r is a partitioned relation, we let $SURE(r)$ be the set of sure tuples for f and let $MAYBE(r)$ be the set of maybe tuples for r. If r is a partitioned relation with scheme R, and s is a partial relation on the same scheme, s *approximates* r, written $s \rhd r$, if $SURE(r) \cup MAYBE(r) \supseteq s \supseteq SURE(r)$. The possibility function we shall use for partitioned relations is

$$POSS_B(r) = \{q \mid q \text{ is a close extension of some } s \text{ such that } s \rhd r\}.$$

In writing partitioned relations, we include a dashed line to separate the sure tuples above from the maybe tuples below.

Example 12.27 Figure 12.25 shows a partitioned relation $r(A\ B\ C)$. The partial relations s_1 and s_2 in Figure 12.26 both approximate r, hence the close extensions q_1 of s_1 and q_2 of s_2 in Figure 12.27 are both in $POSS_B(r)$.

$r(A$	B	$C)$
1	$\perp$	2
$\perp$	3	2
- - -	- - -	- - -
1	3	$\perp$
$\perp$	$\perp$	4

Figure 12.25

$s_1(A$	B	$C)$
1	$\perp$	2
$\perp$	3	2
$\perp$	$\perp$	4

$s_2(A$	B	$C)$
1	$\perp$	2
$\perp$	3	2

Figure 12.26

q_1(A	B	C)
1	3	2
1	5	2
1	3	4
1	5	4

q_2(A	B	C)
1	3	2

Figure 12.27

If r contains only maybe tuples, $POSS_B(r)$ will contain the empty relation along with non-empty relations. Such a situation is not possible for partial relations with any closed possibility function.

We now syntactically characterize strength under $POSS_B$.

Lemma 12.2 For partitioned relations r and s over scheme R and the possibility function $POSS_B$, $r \sqsupseteq s$ if and only if

1. every sure tuple in s is subsumed by a sure tuple of r, and
2. every tuple in r subsumes some tuple in s.

Proof (if) We must show that if the two conditions hold, then $POSS_B(r) \subseteq POSS_B(s)$. Let $\hat{r}$ be a relation in $POSS_B(r)$. In order to show $\hat{r} \in POSS_B(s)$, we must show that every sure tuple in s is subsumed by some tuple in $\hat{r}$ and that every tuple in $\hat{r}$ subsumes some tuple in s. Let t_s be any sure tuple in s. By condition 1, there is a sure tuple $t_r \in r$ such that $t_r \geq t_s$. By the choice of $\hat{r}$, $\hat{r}$ contains a tuple $t_{\hat{r}}$ such that $t_{\hat{r}} \geq t_r$. Therefore, $t_{\hat{r}} \geq t_s$. Now take an arbitrary tuple $u_{\hat{r}}$ in $\hat{r}$. There must be a tuple u_r in r such that $u_{\hat{r}} \geq u_r$. By condition 2, s contains a tuple u_s such that $u_r \geq u_s$, so $u_{\hat{r}} \geq u_s$. Since the choice of $\hat{r}$ was arbitrary, $POSS_B(r) \subseteq POSS_B(s)$ and so $r \sqsupseteq s$.

(only if) Assume $r \sqsupseteq s$. Let $q = SURE(r)$. Form a completion $\hat{q}$ of q by filling in nulls in q with values that appear nowhere in s. By this construction of $\hat{q}$, $\hat{q} \in POSS_B(r)$. Since $r \sqsupseteq s$, $\hat{q}$ is in $POSS_B(s)$, so for any sure tuple $t_s \in s$, q contains a tuple $t_{\hat{q}}$ such that $t_{\hat{q}} \geq t_s$. Tuple $t_{\hat{q}}$ came from a sure tuple t_r in r by using values that do not appear in t_s. Hence, t_r must subsume t_s, and condition 1 is met.

Suppose now $q = SURE(r) \cup MAYBE(r)$. Form a completion $\hat{q}$ of q by filling in nulls using values that do not appear in s. As before, $\hat{q} \in POSS_B(r)$, hence $\hat{q}$ is in $POSS_B(s)$. Take any tuple $t_r \in r$ and let t_q be the tuple in $\hat{q}$ obtained by filling t_r (which appears in q). Since $\hat{q} \in POSS_B(s)$, s contains a tuple t_s such that $t_{\hat{q}} \geq t_s$. Since the filled-in values of $t_{\hat{q}}$ could not match t_s, we must have $t_r \geq t_s$, which fulfills condition 2.

Example 12.28 Let r be the partitioned relation in Figure 12.25 and let s be the partitioned relation shown in Figure 12.28. From Lemma 12.2 we see that $s \sqsupseteq r$, but $r \not\sqsupseteq s$. Indeed, the total relation q_2 in Figure 12.27 is in $POSS_B(r)$ but not in $POSS_B(s)$.

$s(A$	B	$C)$
1	$\perp$	2
$\perp$	3	2
1	3	4
----	----	----
1	$\perp$	4
7	$\perp$	4

Figure 12.28

Lemma 12.2 lets us characterize redundant tuples: those whose removal does not change the strength of a relation.

Corollary Let r be a partitioned relation over scheme R and let t be a tuple in r. Let r' be the partitioned relation formed by removing t from r. Then $r \equiv r'$ if and only if

1. $t \in SURE(r)$, t is subsumed by a sure tuple of r, and t subsumes a sure or maybe tuple of r; or
2. $t \in MAYBE(r)$ and t subsumes a sure or maybe tuple of r.

Proof Left to the reader (see Exercise 12.37).

The second case of the corollary implies that any $t \in SURE(r) \cap MAYBE(r)$ can be removed from $MAYBE(r)$. If r' is formed from r by removing a maybe tuple, then $r' \sqsupseteq r$. If r' is formed by removing a sure tuple, we could have r' and r incomparable in strength.

Example 12.29 Consider the partitioned relations $r(A\ B\ C)$ and $r'(A\ B\ C)$ in Figure 12.29. Neither r nor r' has any maybe tuples; r' is r with a sure tuple removed. Total relation $\hat{r}$ in Figure 12.30 is in $POSS_B(r)$ but not in $POSS_B(r')$; $\hat{r}'$ is in $POSS_B(r')$ but not in $POSS_B(r)$.

$r(A$	B	$C)$
1	$\perp$	2
$\perp$	3	4
----	----	----

$r'(A$	B	$C)$
1	$\perp$	2
----	----	----

Figure 12.29

$\hat{r}$(A	B	C)
1	5	2
6	3	4

$\hat{r}'$(A	B	C)
1	5	2

Figure 12.30

For discussing restricted generalizations, we need to characterize when one partitioned relation is strictly stronger than another. Assume r and s are partitioned relations such that $r \supsetneq s$. There must be a total relation $\hat{s}$ in $POSS_B(s)$ that is not in $POSS_B(r)$. How could s fail to be in $POSS_B(r)$? The first possibility is that r contains a sure tuple t_r that is not subsumed by any tuple in $\hat{s}$. It follows that there is no tuple in s that subsumes t_r. The second possibility is that $\hat{s}$ has a tuple $t_{\hat{s}}$ that does not subsume any tuple in r. It follows that the tuple $t_s \in s$ from which $t_{\hat{s}}$ came does not subsume any tuple in r. To summarize:

Lemma 12.3 Let r and s be partitioned relations such that $r \sqsupseteq s$. If $r \sqsupset s$, then

1. r contains a sure tuple that is not subsumed by any tuple in s, or
2. s contains a tuple that does not subsume any tuple in r.

The converse of Lemma 12.3 is left as Exercise 12.38.

We are ready to consider generalizations of relational operators under $POSS_B$. Projection has a natural generalization. If r is a partitioned relation with scheme R and $X \subseteq R$, we define

$$\pi_X^B(r) = s(X)$$

where

$$SURE(s) = \{t(X) \mid t \text{ in } SURE(r)\} \text{ and}$$
$$MAYBE(s) = \{t(X) \mid t \text{ in } MAYBE(r)\}.$$

Example 12.30 Let $r(A\ B\ C)$ be the partitioned relation in Figure 12.31. Figure 12.32 shows $s = \pi_{BC}^B(r)$. The maybe tuple $\langle \perp\ 4\rangle$ can be removed from s by the corollary to Lemma 12.2.

The proof that π^B is adequate and precise is left as Exercise 12.39. We next generalize select with an equality to constant comparison. Let r be a partitioned relation on scheme R and let $A \in R$. Define

$$\sigma_{A=a}^B(r) = s(R)$$

where

$$\begin{aligned} SURE(s) &= \{t \mid t \in SURE(r) \text{ and } t(A) = a\} \\ MAYBE(s) &= \{t \mid t \in MAYBE(r) \text{ and } t(A) = a\} \cup \\ &\quad \{t' \mid \text{there is a } t \in r \text{ such that } t(R - A) = \\ &\quad\ t'(R - A),\ t(A) = \bot \text{ and } t'(A) = a\}. \end{aligned}$$

Example 12.31 Let r be the partitioned relation in Figure 12.31. Figure 12.33 shows $s = \sigma^B_{A=1}(r)$.

$r(A$	B	$C)$
1	$\bot$	4
$\bot$	5	6
----	----	----
2	$\bot$	4
1	3	6

Figure 12.31

$s(B$	$C)$
$\bot$	4
5	6
----	----
$\bot$	4
3	6

Figure 12.32

$s(A$	B	$C)$
1	$\bot$	4
----	----	----
1	5	6
1	3	6

Figure 12.33

Lemma 12.4 The operator $\sigma^B_{A=a}$ is a precise generalization of $\sigma_{A=a}$ for $POSS_B$.

Proof Let $s = \sigma^B_{A=a}(r)$ for an arbitrary partitioned relation $r(R)$ with $A \in R$. First, we show $POSS_B(s) \supseteq \sigma_{A=a}(POSS_B(r))$. Let q be in $\sigma_{A=a}(POSS_B(r))$.

Let $\hat{r}$ be a relation in $POSS_B(r)$ such that $q = \sigma_{A=a}(\hat{r})$. We must show $q \in POSS_B(s)$. Let t be in $SURE(s)$. We must exhibit a tuple t_q of q such that $t_q \geq t_s$. Since $t_s \in SURE(s)$, $t_s(A) = a$ and $t_s \in SURE(r)$. Hence, $\hat{r}$ has a tuple $t_r \geq t_s$. We see that $t_{\hat{r}}(A)=a$, so $t_{\hat{r}} \in q$ and is the desired tuple t_q above. We next must show that for any tuple $t_q \in q$ there is a tuple $t_s \in s$ such that $t_q \geq t_s$. We know $t_q \in \hat{r}$, so there is a tuple $t_r \in r$ such that $t_q \geq t_r$. Either $t_r(A) = a$ or $t_r(A) = \perp$. In the former case, $t_r \in s$. In the latter case, $t_r' \in s$, where $t_r'(A) = a$ and $t_r'(R - A) = t_r(R - A)$. Since $t_q \geq t_r$, and $t_q(A) = a$, $t_q \geq t_r'$. We conclude $q \in POSS_B(s)$, as desired.

Second, we show $POSS_B(s) \subseteq \sigma_{A=a}(POSS_B(r))$. Let $q \in POSS_B(s)$. We must find $\hat{r}$ in $POSS_B(r)$ such that $q = \sigma_{A=a}(\hat{r})$. Note that $q = \sigma_{A=a}(q)$. If $q \notin POSS_B(r)$, it can only be because r contains a sure tuple t_r that is not subsumed by any tuple in q. Let $\hat{r}$ include q and a total tuple t_r' for every sure tuple t_r in r not covered by q. Choose t_r' so that $t_r'(A) \neq a$. We have $\hat{r} \in POSS_B(r)$, $q = \sigma_{A=a}(r)$, hence $q \in \sigma_{A=a}(POSS_B(r))$. We have shown $POSS_B(s) = \sigma_{A=a}(POSS_B(r))$, so $\sigma^B_{A=a}$ is precise.

We leave it to the reader to generalize $\sigma_{A=B}$ for $POSS_B$.

Let r and s both be partitioned relations on scheme r. The generalized union given by

$$r \cup^B s = q(R)$$

where

$$SURE(q) = SURE(r) \cup SURE(s) \text{ and}$$
$$MAYBE(q) = MAYBE(r) \cup MAYBE(s)$$

is precise (see Exercise 12.42).

Finally, we generalize join for $POSS_B$. Let $r(R)$ and $s(S)$ be partitioned relations where $R \cap S = X$. Say that $t_r \in r$ and $t_s \in s$ are *compatible on* X if for every $A \in X$, $t_r(A) = t_s(A)$ or at least one of $t_r(A)$ and $t_s(A)$ is null. Define

$$r \bowtie^B s = q(R\,S)$$

where

$$SURE(q) = \{t(RS) | \text{there are } t_r \in SURE(r), t_s \in (SURE(s) \text{ such that } t_r(X){\downarrow}, t_s(X){\downarrow}, t(R) = t_r \text{ and } t(S) = t_s\} \text{ and}$$

$MAYBE(q) = \{t(RS) \mid$ there are compatible tuples $t_r \in r$ and $t_s \in s$ such that $t(R-X) = t_r(R-X)$, $t(S-X) = t_s(R-X)$, and for each $A \in X$, if $t_r(A)\downarrow$, then $t(A) = t_r(A)$, else $t(A) = t_s(A)\}$.

The "*MAYBE*" part of the definitions says we join compatible tuples by following the more definite tuple on each attribute in X. This definition makes $SURE(q)$ a subset of $MAYBE(q)$, but we may drop the duplicate tuples from $MAYBE(q)$.

Example 12.32 Let $r(A\ B)$ and $s(B\ C)$ be the partitioned relations in Figure 12.34. Figure 12.35 shows $q = r \bowtie^B s$.

r(A	B)
⊥	1
3	⊥
- - -	- - -
4	1

s(B	C)
1	2
- - -	- - -
5	2
⊥	6

Figure 12.34

q(A	B	C)
1	1	2
- - -	- - -	- - -
⊥	1	6
3	1	2
3	5	6
3	⊥	6
4	1	2
4	1	6

Figure 12.35

This generalization of join is not precise. Referring to the last example, any relation in $POSS_B(r) \bowtie POSS_B(s)$ satisfies the JD $*[AB, BC]$. There are total relations in $POSS_B(q)$ that do not satisfy this JD.

Lemma 12.5 The operator $\bowtie^B$ is an adequate and restricted generalization of $\bowtie$ for $POSS_B$.

Proof For the proof, let $r(R)$ and $s(S)$ be partitioned relations, let $X = R \cap S$ and let $q = r \bowtie^B s$.

($\bowtie^B$ adequate) Let $\hat{r} \in POSS_B(r)$ and $\hat{s} \in POSS_B(s)$. Let $\hat{q} = \hat{r} \bowtie \hat{s}$. We must show $\hat{q} \in POSS_B(q)$. Let t_q be any tuple in $SURE(q)$. There must be tuples $t_r \in r$ and $t_s \in s$ that join to form t_q, and, furthermore, $t_r(X)\downarrow$, $t_s(X)\downarrow$ and $t_r(X) = t_s(X)$. By the choice of $\hat{r}$, it contains a tuple $t_{\hat{r}}$ such that $t_{\hat{r}} \geq t_r$. Likewise, s contains a tuple $t_{\hat{s}}$ with $t_{\hat{s}} \geq t_s$. Let $t_{\hat{q}} = t_{\hat{r}} \bowtie t_{\hat{s}}$. We have $t_{\hat{q}} \geq t_q$. Suppose now that u_q is an arbitrary tuple from $\hat{q}$. This tuple must be $u_{\hat{r}} \bowtie u_{\hat{s}}$ for some $u_{\hat{r}} \in \hat{r}$ and $u_{\hat{s}} \in \hat{s}$. In turn, r must contain a u_r such that $u_{\hat{r}} \geq u_r$, and $s \in u_s$ such that $u_{\hat{s}} \geq u_s$. Tuples u_r and u_s must becompatible and q therefore contains a maybe tuple u_q constructed from u_r and u_s. It follows fairly directly that $u_q \geq u_{\hat{q}}$. Thus, q is in $POSS_B(q)$ and so $POSS_B(q) \supseteq POSS_B(r) \bowtie POSS_B(s)$, as desired.

($\bowtie^B$ restricted) Suppose q' is a partitioned relation such that $q' \supsetneq q$ and $POSS_B(q) \supseteq POSS_B(r) \bowtie POSS_B(s)$. Lemma 12.3 provides two cases to consider.

Case 1 $SURE(q')$ contains a tuple t_q' that is not subsumed by any tuple in q. We make the following claim.

Claim Any completion $\hat{q}$ of $SURE(q)$ is in $POSS_B(r) \bowtie POSS_B(s)$.

The proof of the claim is left as Exercise 12.42. Form a completion $\hat{q}$ of $SURE(q)$ using values not in q' to fill in nulls in $SURE(q)$. By its construction, no tuple in $\hat{q}$ subsumes tuple t_q' in q', so $\hat{q} \notin POSS_B(q')$. By the claim, we have a contradiction to $POSS_B(q') \supseteq POSS_B(r)$.

Case 2 Relation q contains a tuple t_q that does not subsume any tuple in q'. Consider the tuple $t_q \downarrow\geq t_q$ that is obtained by filling in t_q with values not in q'. No tuple in q' is subsumed by $t_{\hat{q}}$. By the construction of $\hat{t}_q$, r must contain a tuple t_r such that $\hat{t}_q(R) \downarrow\geq t_r$, and s must contain t_s where $\hat{t}_q(S) \downarrow\geq t_s$. Choose $\hat{r} \in POSS_B(r)$ that contains $\hat{t}_q(R)$ and choose $\hat{s} \in POSS_B(s)$ that contains $\hat{t}_q(S)$. We see $\hat{r} \bowtie \hat{s}$ contains $\hat{t}_q$, so $\hat{r} \bowtie \hat{s} \notin POSS_B(q')$, a contradiction. We conclude q' as described cannot exist, so $\bowtie^B$ is restricted.

In this section we have considered generalizing relational operators relative to various possibility functions. Only by introducing partitioned relations were we able to obtain a closed possibility function with adequate and restricted generalizations of union, select, project, and join. There may yet exist a closed possibility function for partial relations that admits such generalizations. It is also plausible that incorporating marked nulls would allow precise generalizations where only adequate and restricted ones are achievable now.

12.5 PARTIAL INFORMATION AND DATABASE SEMANTICS

We now turn to handling "subtuples" over a relation scheme. At times it would be useful to view a relation as a set of inhomogeneous tuples. Different tuples could have different schemes.

Example 12.33 Consider the attributes ADVISOR, DEPT, and STUDENT. We might want to store a tuple $\langle a\ d \rangle$ over ADVISOR DEPT, meaning a does advising for department d. We might also want a tuple $\langle d\ s \rangle$ over DEPT STUDENT, representing that d is the major department for s. We also might want a tuple $\langle a\ d\ s \rangle$ over all three attributes, meaning a is the advisor of s for department d. Possibly, we may want tuples over STUDENT alone, simply to record all the students.

The problems in handling a relation with inhomogeneous tuples are similar to those in treating a database as a single semantic entity. We might view a database as a single relation with inhomogeneous tuples. In Chapter 9, we were interested in databases that represented instances over a universal scheme, although we saw there that a database can have states that represent no universal instance. There is no efficient algorithm known to test if a given database state is the projection of a common instance. (In Chapter 13 we explore a class of database schemes for which such a test can be done efficiently.) It may be too restrictive to insist a database represent a single relation. If we do not adopt that view, though, how can we treat a database as a unit and how do we discuss enforcing constraints on the database as a whole?

12.5.1 Universal Relation Assumptions

The requirement that the relations in a database all be projections of some common instance is called the *universal instance assumption* (UIA). A database is seen as representing a single relation over a universal scheme—usually the join of all the relations in the database. The UIA is often a reasonable assumption for design purposes, even if actual states of the database will not always conform to it. It is particularly apt when the design starts with a single scheme. We can draw a broad analogy between the UIA and the assumption that a programming language can be described by a context-free grammar. While the basic syntax of the language may be described by a set of productions, the semantic actions associated with those productions may actually make certain constructs context-sensitive. Nevertheless, the context-free grammar is a useful tool for organizing a compiler for the language.

Why prefer a database over a single relation? First, we may eliminate redundancy going from the single relation to the database. Second, we may find it useful to store states of the database that do not correspond to any single relation.

Example 12.34 We could represent the history relation of Table 12.1 as a database of two relations: *info*(EMPLOYEE SALARY) and *pastinfo*(EMPLOYEE PREVEMP PREVJOB PREVSAL). We could then store a tuple ⟨Lombardi 13,200⟩ in *info* without storing a tuple for Lombardi in *pastinfo*, in the case that Lombardi had no previous job. The tuple ⟨Lombardi 13,200⟩ in *info* would *not* have the same meaning as the tuple ⟨Lombardi 13,200 $\perp$ $\perp$ $\perp$⟩ in *history*, since the latter asserts Lombardi had a previous job, but we do not know the details.

There are situations where the UIA does not make sense, even at the design level. The design process does not necessarily start from a single relation scheme. The database can include facts on a wide range of subjects, where it is not meaningful or natural to discuss a tuple that ranges over the entire set of attributes. The design process can start with a set of relation schemes, and proceed by combining and decomposing those schemes. We do not want the UIA, but we do want some consistency assumption about the meaning of attributes in different schemes. If in one scheme DATE means the birthdate of an employee, and somewhere else it means the date of appointment, we have semantic problems with combining relations using natural join. If we assume that two occurrences of the same attribute in different schemes always have different meanings we can miss important connections.

The *universal relation scheme assumption* (URSA) on a database scheme requires an attribute mean the same everywhere it appears, that it represents a single role of a class of entities. DATE, as used above, represented different roles for the class of dates. URSA is always satisfied when a database scheme is obtained from a single relation scheme by decomposition. A consequence of URSA is that tuples over a given set of attributes have a single meaning. Hence, an URSA database should never contain two relations over the same scheme, for both relations would contain the same set of tuples. In an URSA database, the scheme is sufficient to identify a relation. A set of attributes determines a unique semantic connection among them, if any connection at all exists. To contrast the UIA and URSA, the UIA requires a "universal extension," while URSA only requires "universal intension."

To satisfy URSA, it may be necessary to rename attributes.

Example 12.35 Suppose we start with a relation *advises*(FACULTY STUDENT) to record advisors of students and a relation *teaches*(FACULTY COURSE STUDENT) to record students' instructors and courses. URSA is violated because of the two connections between FACULTY and STUDENT. To satisfy URSA, we can rename the two occurrences of FACULTY to ADVISOR and INSTRUCTOR, to reflect the different roles involved. Some information is lost here—that ADVISOR and INSTRUCTOR are different roles for the same class of entities. This connection may be discernable from the domains involved. It can be recaptured explicitly in a generalization hierarchy, which relates the different roles of a class of entities.

12.5.2 Placeholders and Subscheme Relations

We return to the problem of inhomogeneous tuples in a single relation. In Example 12.34 we saw that decomposing a relation into database let us represent facts over a subscheme of the original relation. Decomposition is not always the answer.

Example 12.36 Referring back to Example 12.33, we wanted to represent tuples over the schemes ADVISOR DEPT, DEPT STUDENT and ADVISOR DEPT STUDENT. Using a database with two relations, r_1(ADVISOR DEPT) and r_2(DEPT STUDENT), is not sufficient. We have no way to represent tuples over ADVISOR DEPT STUDENT. We cannot use the join of r_1 and r_2, since an advisor advising for a department and a student majoring in the same department do not necessarily imply that advisor advises that student.

One approach is to introduce a new special symbol, called a *placeholder*, to pad out a tuple over a subscheme to the entire relation scheme. We use a dash as the placeholder symbol. We treat a padded tuple as a tuple over the subscheme of attributes where the placeholder does not appear.

Example 12.37 Table 12.11 shows a relation r(ADVISOR DEPT STUDENT) with subtuples represented using placeholders. The tuple t_2 =

⟨Thorton Math -⟩ is interpreted as a regular tuple over the scheme ADVISOR DEPT. Tuple t_2 states only that Thorton does advising for the Math department. There is no assertion that Thorton actually advises some students, which the tuple ⟨Thorton Math-⟩ makes.

Table 12.11 Relation r.

	r(ADVISOR	DEPT	STUDENT)
t_1	Thomas	Math	Walker
t_2	Thorton	Math	-
t_3	-	Econ	Wilson
t_4	-	-	Wu

While nulls and placeholders can both appear in a relation, we restrict ourselves to just placeholders for the moment. As with nulls, we may want to control the use of placeholders in a relation. We may allow subtuples only over certain schemes. In the last example, we might disallow subtuples over just ADVISOR and STUDENT. We could use a formalism similar to existence constraints to restrict the legal schemes for subtuples, but it seems more natural here to simply list the allowable schemes for subtuples. We call an allowable scheme for a subtuple an *object*. Objects are basically those sets of attributes over which tuples "make sense." We constrain a relation $r(R)$ by giving a set of objects $\mathbf{O} = \{W_1, W_2, \ldots, W_n\}$, where $W_i \subseteq R$, $1 \leq i \leq n$. Relation r *satisfies* $\mathbf{O}$ if for any tuple t in r, the set of attributes where placeholders appear for t is $R - W_i$ for some object $W_i \in \mathbf{O}$.

Example 12.38 Relation r in Table 12.11 satisfies the set of objects $\mathbf{O}$ = {ADVISOR DEPT STUDENT, ADVISOR DEPT, DEPT STUDENT, DEPT, STUDENT}.

While placeholders are adequate for representing subtuples in a single relation, we use another method when working with databases. Given a set of objects $\mathbf{O}$, we represent subtuples in a database that has one relation for each object $W \in \mathbf{O}$. Naturally, the relation on W contains all the subtuples with scheme W. That is, we use $\mathbf{O}$ as a database scheme. We are departing from our usual practice. Up to now, we never had two relations $r(R)$ and $s(S)$ in a database if $R \subseteq S$. Under the UIA, there is no point in such an arrangement, since $r = \pi_R(s)$. With just URSA, however, the arrangement makes sense.

The use of separate relations for subtuples also avoids a consistency problem. If placeholders are used, a given subtuple could properly appear in multiple relations. A tuple $\langle b\ c \rangle$ over $B\ C$ could be padded to appear in a relation $r(A\ B\ C)$ or a relation $s(B\ C\ D)$. To obey URSA, it seems that if a BC-tuple appears in r, it should also appear in s. When inserting a subtuple into a relation, we must test to see if that subtuple could properly be included in another relation. Using a separate relation for each type of subtuple removes the need for such cross-checking.

Example 12.39 Consider the set of objects $\mathbf{O}$ = {ADVISOR DEPT, DEPT STUDENT, ADVISOR DEPT STUDENT, DEPT STUDENT MATRIC}. MATRIC is the date when a student reaches upper-division standing in a department. We could store tuples over these objects in two relations, r_1(ADVISOR DEPT STUDENT) and r_2(DEPT STUDENT MATRIC), using placeholders. It is necessary to check that a DEPT STUDENT-tuple inserted into one relation is also inserted into the other. With one relation per object, no checking is necessary.

12.5.3 Database Semantics and Window Functions

In this and subsequent sections we consider treating a database as a semantic unit. We allow that a database contain relations on schemes that are subschemes of other relations. To be consistent with URSA, there are restrictions on such relations. Let $r(R)$ and $s(S)$ be relations in the same database, where $R \subseteq S$. By URSA, a set of attributes uniquely determines a semantic connection among themselves. Since R is a subset of S, whatever the connection among the attributes of R, it must be an aspect of the connection among the attributes in S. Thus, if t is a tuple of s, $t(R)$ should be a tuple in r. Equivalently, r should contain $\pi_R(s)$. This constraint is the *containment condition on objects*. While the containment condition seems to require storing much redundant information, it is easy to think of implementations that remove the redundancy.

Example 12.40 We consider a database where the containment condition is not met, and see how URSA is violated. Consider a database of just two relations, r(DEPT STUDENT) and s(ADVISOR DEPT STUDENT.) Suppose a tuple $\langle d\ s \rangle \in r$ means student s is taking a course from department d, and a tuple $\langle a\ d\ s \rangle \in s$ means a advises s for department d. The containment condition will be violated if we allow that a student can receive advising from a department before taking any courses offered by the department. The connection between DEPT and STUDENT given by r does not agree with the connection implied by s. At least one of DEPT or STUDENT must represent different roles in r and s, so URSA is violated.

We make a brief detour to consider the containment condition in light of updates. For a database, it is usual to restrict the set of users who can update a particular relation. Suppose a database contains relations $r(R)$ and $s(S)$ where $R \subseteq S$. At first it may seem permission to update s should imply permission to update r. Actually, it is quite reasonable to give update permission for s without permission for r. Having update permission for r means one can constrain updates to s, since a tuple t cannot be added to s unless $t(R) \in r$.

Example 12.41 Consider a database d on the set of objects {DEPT, DEPT COURSE#, DEPT COURSE SEMESTER YEAR, STUDENT, DEPT COURSE# SEMESTER YEAR STUDENT}. By the containment condition, a tuple ⟨Econ 101 Spring 1980⟩ over DEPT COURSE# SEMESTER YEAR could only be added to the database if there is already a tuple ⟨Econ 101⟩ over DEPT COURSE#. That is, a course cannot be scheduled for a given semester unless the course actually exists. Typically, the president of the university has authority to authorize an insertion of a DEPT-tuple (creating a new department). An academic vice-president or dean can authorize insertions over DEPT COURSE# (creating new courses). The admissions office authorizes STUDENT-tuples (admitting new students). A department chairman can add tuples over DEPT COURSE# SEMESTER YEAR (deciding course offerings). A student authorizes tuples over DEPT COURSE# SEMESTER YEAR STUDENT (enrolling in a course).

We return to the ramifications of URSA in a database. There are connections among attributes that are not captured directly by any relation in a database. Rather, the connections are realized by combining two or more relations. Such derived connections should be consistent with the connection embodied explicitly in relations.

Example 12.42 Consider a database d with relations r_1(FACULTY CLASS), giving the classes a faculty member teaches, and r_2(CLASS ROOM), giving the room where a class meets. It seems reasonable to connect FACULTY, CLASS, and ROOM by joining r_1 and r_2. Suppose, however, there is also a relation r_3(FACULTY ROOM) giving offices for faculty members. There is a disagreement between the explicit connection of FACULTY and ROOM in r_3 and the derived connection given by $r_1 \bowtie r_2$. Considered individually, it is not immediately apparent that r_1, r_2, and r_3 violate URSA. It is the derived connection from $r_1 \bowtie r_2$ that causes problems.

Whatever implicit connections among attributes we derive from a database should be consistent with each other and the explicit connections given in relations. We extend the meaning of *object* to include the schemes of derived relations on a database. The meanings associated with objects must be consistent, whether the objects correspond to stored or derived relations in a database.

To afford a uniform view of objects, stored and derived, we use a *window function*, so called because it gives a consistent set of views of the database, one for each object. A window function maps an object and a database to the relation, stored or derived, that the database assigns to that object. We denote the value of a window function on object W and database d by $[W, d]$, or simply $[W]$ where d is understood. Thus, $[W, d]$ is always a relation with scheme W, whose value depends on the state of d.

A window function allows no room for multiple meanings for objects, since it returns a unique relation for any object. Unique meanings for objects is certainly dictated by URSA, but meanings for different objects must be consistent. Hence, we make a *containment condition for windows*: If W and Z are objects for a database d, and $W \subseteq Z$, then $[W, d] \supseteq \pi_W([Z, d])$.

A window function can easily be extended to map every set of attributes to a relation over those attributes. Let X be a set of attributes and let $\mathbf{O}$ be a set of objects for a database d. We let

$$[X, d] = \bigcup_{\substack{W \in \mathbf{O} \\ W \supseteq X}} \pi_X([W, d]).$$

The meaning for a set of attributes that is not an object comes from the meaning of objects containing that set. If the original window function satisfies the containment condition, then the extended window function satisfies the containment condition, and agrees with the original function on any object (see Exercise 12.45). If a set of attributes X is not contained in any object in $\mathbf{O}$, $[X, d]$ will always be the empty relation. The database assigns no meaning to the attributes in X taken as a group.

Some of the window functions we consider are defined directly for arbitrary sets of attributes. They do not make use of objects besides those corresponding to stored relations. Such a definition gives rise implicitly to a set of objects, however. From the containment condition, for any set of attributes X,

$$[X] \supseteq \bigcup_{Y \supsetneq X} \pi_X([Y]).$$

For some sets X, the containment will always be equality. For other X, there will be states of the database for which the containment is strict. Such an X is an *implicit object* of the window.

We consider one simple window function, and a problem with it, which will motivate another condition on window functions. Let $\mathbf{U}$ be the set of all attributes appearing in the schemes of a database d. We define a window function with $\mathbf{U}$ as the only object. Let $[\mathbf{U}, d]_1 = \bowtie d$. (The subscript will distinguish this particular window function from others.) This window function treats the database as representing a single relation over scheme $\mathbf{U}$. Extending this function to arbitrary sets of attributes we get

$$[X]_1 = \pi_X[\mathbf{U}]_1.$$

The problem with $[\cdot]_1$ is that for a relation $r(R)$ in d, $[R]_1$ does not necessarily equal r, although it is always contained in r. The meaning the window function gives r is not the one given in the database. While the interpretation of a database given by $[\cdot]_1$ satisfies URSA, and the database itself may satisfy URSA, considered together they violate URSA.

In an URSA database, no two stored relations may have the same scheme, so we may use schemes alone to distinguish relations. If R is a relation scheme, we use $\bar{r}(R)$ to denote the database relation with scheme R, whatever its actual name. We impose a *faithfulness condition* on window functions: for any relation scheme R of a database d, $[R, d] = \bar{r}(R)$. The window function $[\cdot]_1$ violates the faithfulness condition. Unless constraints are imposed on the database state, the faithfulness condition means relation schemes are always implicit objects of window functions.

In subsequent sections we examine several window functions. The first is defined using joins of relations, but not all the relations in the database at once. The others are defined as projections from a single relation, albeit one with nulls.

12.5.4 A Window Function Based on Joins

We assume a set of objects $\mathbf{O}$, of which a subset, $\mathbf{R}$, constitute the database scheme for a database d. As before, we let $\bar{r}(R)$ denote the relation on R for any $R \in \mathbf{R}$. We allow that one relation scheme be a subscheme of another, but require that the relations involved satisfy the containment condition. Because of the nature of the window function we shall define, we also require that every object W in $\mathbf{O}$ be the union of objects (schemes) in $\mathbf{R}$. We define a window function

$$[W, d]_{\mathbf{R},\mathbf{O}} = \mathop{\bowtie}_{\substack{R \in \mathbf{R} \\ R \subseteq W}} \bar{r}(R), \text{ for } W \in \mathbf{O}.$$

The relation for an object is the join of all stored relations whose schemes are contained in the object.

Example 12.43 Consider the set of objects $\mathbf{O}$ = {DEPT, DEPT COURSE#, DEPT COURSE# INSTRUCTOR, DEPT COURSE# CREDITS, DEPT COURSE# STUDENT, DEPT COURSE# STUDENT CREDITS, DEPT COURSE# STUDENT INSTRUCTOR} and a database d over the subset $\mathbf{R}$ = {DEPT, DEPT COURSE#, DEPT COURSE# INSTRUCTOR, DEPT COURSE# CREDITS, DEPT COURSE# STUDENT}. The sets $\mathbf{R}$ and $\mathbf{O}$ are pictured in Figure 12.36. Solid lines indicate schemes in R; dashed lines indicate objects in $\mathbf{O} - \mathbf{R}$. Table 12.12 gives a state of d. Table 12.13 gives the values of [DEPT COURSE# INSTRUCTOR STUDENT]$_{\mathbf{R},\mathbf{O}}$ and [DEPT COURSE# CREDITS STUDENT]$_{\mathbf{R},\mathbf{O}}$.

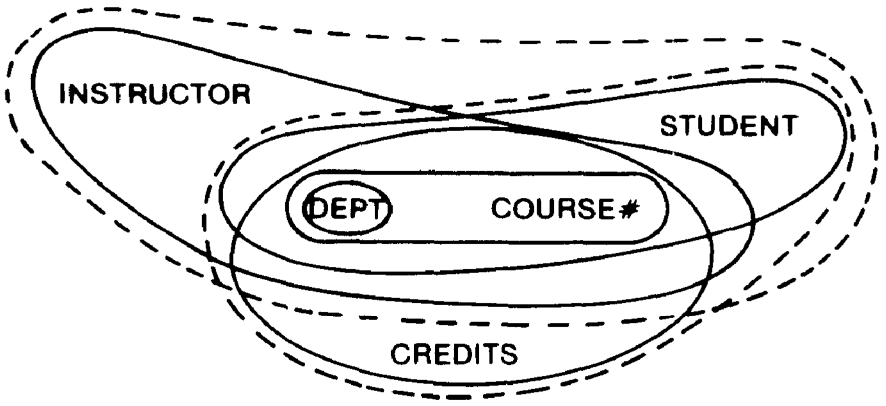

Figure 12.36

The window function $[\cdot]_{\mathbf{R},\mathbf{O}}$ satisfies the containment condition, even if the database d does not (see Exercise 12.47). It is also possible to prune the join needed for some objects (see Exercise 12.48). This window function is faithful, provided the database satisfies the containment condition. In proof, note that for schemes R and S in $\mathbf{R}$, if $R \subseteq S$, then $\bar{r}(R) \bowtie \bar{r}(S) = \bar{r}(S)$. Thus, for any $S \in \mathbf{R}$

$$\mathop{\bowtie}_{\substack{R \in \mathbf{R} \\ R \subseteq S}} \bar{r}(R) = \bar{r}(S).$$

Table 12.12 State of *d*.

$\bar{r}$(DEPT)	$\bar{r}$(DEPT	COURSE#)
Econ	Econ	101
Math	Econ	102
History	Math	120
English	History	306
	History	308

$\bar{r}$(DEPT	COURSE#	INSTRUCTOR)
Econ	101	Galler
Econ	102	Garvey
Math	120	George
History	306	Gunther

$\bar{r}$(DEPT	COURSE#	CREDITS)
Econ	101	4
Econ	102	3
Math	120	4
History	306	4
History	308	3

$\bar{r}$(DEPT	COURSE#	STUDENT)
Econ	101	Adams
Econ	101	Allen
Econ	102	Andrews
History	308	Adams
History	308	Andrews

We extend $[\cdot]_{R,O}$ to arbitrary sets of attributes as indicated in the last section.

Example 12.44 Continuing Example 12.43, $[\text{STUDENT CREDITS}]_{R,O}$ and $[\text{STUDENT INSTRUCTOR}]_{R,O}$ are shown in Table 12.14. The value of $[\text{INSTRUCTOR CREDITS}]_{R,O}$ will always be empty because no object in **O** contains both INSTRUCTOR and CREDITS.

We shall return to this join-based window function when we examine the PIQUE query language in Chapter 15.

Table 12.13 Values of Windows.

[DEPT	COURSE#	INSTRUCTOR	STUDENT]$_{R,O}$
Econ	101	Galler	Adams
Econ	101	Galler	Allen
Econ	102	Garvey	Andrews

[DEPT	COURSE#	CREDITS	STUDENT]$_{R,O}$
Econ	101	4	Adams
Econ	101	4	Allen
Econ	102	3	Andrews
History	308	3	Adams
History	308	3	Andrews

Table 12.14 Values of Windows (continued)

[STUDENT	CREDITS]$_{R,O}$
Adams	4
Adams	3
Allen	4
Andrews	3
Andrews	3

[STUDENT	INSTRUCTOR]$_{R,O}$
Adams	Galler
Allen	Galler
Andrews	Garvey

12.5.5 Weak Instances

The underlying concept for the next definitions of window functions arose from the problems of enforcing data dependencies globally on a database.

Example 12.45 Consider the database in Table 12.15, with relations on the schemes DEPT COURSE# CREDITS SEMESTER and DEPT COURSES# CREDITS STUDENT. Both relations satisfy the FD DEPT COURSE# $\rightarrow$ CREDITS, but they represent different functions from DEPT COURSE# to CREDITS.

While a database state need not be the projection of a common instance, every database state is in a sense contained in a universal instance.

Definition 12.11 Let d be a database over the scheme $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$, where $\mathbf{U} = R_1 R_2 \cdots R_p$. A relation s over scheme $\mathbf{U}$ is a *weak instance* for d if

$$\pi_{R_i}(s) \supseteq \bar{r}(R_i)$$

for every scheme $R_i \in \mathbf{R}$.

Table 12.15 Inconsistent Database.

$\bar{r}$(DEPT	COURSE#	CREDITS	SEMESTER)
Econ	106	4	Spring
Econ	108	4	Fall
Math	211	3	Fall
Math	211	3	Spring
Math	286	3	Fall

$\bar{r}$(DEPT	COURSE#	CREDITS	STUDENTS)
Econ	106	4	Balfour
Econ	106	4	Berents
Econ	108	3	Balfour
Math	286	4	Berents
Math	286	4	Brown

The view here is that a database stands for its set of weak instances. Using weak instances to discuss a database does assume that a tuple over the universal scheme **U** makes sense, if not a universal instance over that scheme. We can use weak instances to discuss applying data dependencies globally to a database.

Definition 12.12 Let d be a database over the database scheme $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$, where $\mathbf{U} = R_1 R_2 \cdots R_p$. Let **C** be a set of dependencies over **U**. Database d *globally satisfies* **C**, if d has a weak instance that satisfies **C** in the usual sense.

Example 12.46 If d is the database of Example 12.45, and $F = \{$DEPT COURSE# $\rightarrow$ CREDITS$\}$, then d does not globally satisfy F. If we changed $\bar{r}$(DEPT COURSE# CREDITS STUDENT) to be as shown in Table 12.16, then d will globally satisfy F. A weak instance s of d satisfying F is shown in Table 12.17.

Table 12.16 Modified Relation.

$\bar{r}$(DEPT	COURSE#	CREDITS	STUDENT)
Econ	106	4	Balfour
Econ	106	4	Berents
Econ	108	4	Balfour
Math	286	3	Berents
Math	286	3	Brown

Table 12.17 Weak Instance *s*.

s(DEPT	COURSE#	CREDITS	SEMESTER	STUDENT)
Econ	106	4	Spring	Balfour
Econ	106	4	Spring	Berents
Econ	108	4	Fall	Balfour
Math	211	3	Fall	Butcher
Math	211	3	Spring	Butcher
Math	286	3	Fall	Berents
Math	286	3	Fall	Brown

For a database d over **U** and a set **C** of dependencies on **U**, a relation $s(\mathbf{U})$ is a **C**-*weak instance*, (**C**-WI) for d if s is a weak instance for d satisfying **C**. Evidently, d has a **C**-WI if and only if d globally satisfies **C**.

We can define window functions using representative instances. We look for the common parts of all representative instances for a database. Let d be a database over **U**, let **C** be a set of dependencies over **U**, and let X be an arbitrary subset of **U**. We define

$$[X,d]_{\mathbf{C}} = \bigcap_{\substack{s \text{ a } \mathbf{C}\text{-WI} \\ \text{for } d}} \pi_X(s).$$

That is, $[X,d]_{\mathbf{C}}$ consists of the X-values that appear in some tuple of every **C**-WI for d. This definition satisfies the containment condition. For sets X and Y where $X \subseteq Y$, if t is a tuple that is in $\pi_Y(s)$ for every **C**-WI s for d, then $t(X)$ is surely in $\pi_X(s)$ for every such s. The definition of $[\cdot]_{\mathbf{C}}$ does not immediately give rise to an effective procedure for its evaluation. A database d could have an infinite number of **C**-WIs. Fortunately, when **C** is FDs and JDs, *nchase* can be used to compute $[\cdot]_{\mathbf{C}}$.

Definition 12.13 Let r be a relation in $Rel\uparrow(R)$ and let $X \subseteq R$. The *X-total projection of* r, denoted $\pi\downarrow_X(r)$, is $\{t(X) | t \in r \text{ and } t(X)\downarrow\}$. The X-total projection of r is the X portion of all tuples in r that are definite on X.

If r is a relation on scheme R and $\mathbf{U}$ is a set of attributes containing R, $PAD(r,\mathbf{U})$ is the relation s in $Rel\uparrow(\mathbf{U})$ obtained by padding out each tuple in r to have scheme $\mathbf{U}$. The padding is done with distinct marked nulls. (The relations we shall be padding have no nulls of their own.) For a database d over $\mathbf{U}$, $PAD(d)$ is $\bigcup_{r \in d} PAD(r,\mathbf{U})$. We pad out all the relations in d to a common scheme.

Example 12.47 Let d be the database in Figure 12.37. $PAD(d)$ is the relation s shown in Figure 12.38.

$\bar{r}(A$	$B)$
1	4
1	5
2	4
2	6

$\bar{r}(B$	$C)$
4	7
5	7
6	8

$\bar{r}(A$	$C)$
1	7
2	8
3	8
3	9

Figure 12.37

$s(A$	B	$C\,)$
1	4	$\perp_1$
1	5	$\perp_2$
2	4	$\perp_3$
2	6	$\perp_4$
$\perp_5$	4	7
$\perp_6$	5	7
$\perp_7$	6	8
1	$\perp_8$	7
2	$\perp_9$	8
3	$\perp_{10}$	8
3	$\perp_{11}$	9

Figure 12.38

Nchase from Section 12.2 can be extended to include an analogue of the J-rule in regular chase computation. There is no hard violation of JDs, however, as there was for FDs. The extension is straightforward.

Theorem 12.1 Let d be a database over $\mathbf{U}$ and let $\mathbf{C}$ be a set of FDs and JDs over $\mathbf{U}$. For any subset X of $\mathbf{U}$,

$$[X]_{\mathbf{C}} = \pi\downarrow_X(nchase_{\mathbf{C}}(PAD(d))).$$

Proof Left to the reader (see Exercise 12.56).

Different choices for $\mathbf{C}$ give different flavors of window functions. One possibility is to simply let $\mathbf{C} = \{*[R_1, R_2, \ldots, R_p]\}$ where $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ is the database scheme. For this choice of $\mathbf{C}$, we use $[\cdot]_{*\mathbf{R}}$ to denote the window function, rather than $[\cdot]_{\mathbf{C}}$.

Example 12.48 Let d be the database of Figure 12.37, for which $\mathbf{R} = \{AB, BC, AC\}$. $Nchase_J(s)$ is shown in Figure 12.39, where $J = \{*[\mathbf{R}]\}$ and $s = PAD(d)$. Figure 12.40 shows $[A\ B\ C]_{*\mathbf{R}}$.

$nchase_J(s)$(A	B	C)
1	4	$\perp_1$
1	5	$\perp_2$
2	4	$\perp_3$
2	6	$\perp_4$
$\perp_5$	4	7
$\perp_6$	5	7
$\perp_7$	6	8
1	$\perp_8$	7
2	$\perp_9$	8
3	$\perp_{10}$	8
3	$\perp_{11}$	9
1	4	7
2	6	8

Figure 12.39

[A	B	C] $_{*\mathbf{R}}$
1	4	7
2	6	8

Figure 12.40

The window function $[\cdot]_{*\mathbf{R}}$ is faithful as long as the database d satisfies the containment condition (see Exercise 12.57). For an arbitrary database scheme $\mathbf{R}$, $[\cdot]_{*\mathbf{R}}$ is probably hard to compute, since given a set of attributes X

and a tuple t over X, determining if $t \in [X]_{*\mathbf{R}}$ is NP-complete. In Chapter 13 we shall see a class of database schemes for which $[\cdot]_{*\mathbf{R}}$ can be readily evaluated.

There is a problem in using $[\cdot]_{\mathbf{C}}$ if $\mathbf{C}$ includes a set F of FDs. The database d may not globally satisfy F, hence there are no **C**-WIs. If we plan to use F to define a window function, we want to constrain d to globally satisfy F. Enforcing F on the relations individually is not sufficient, as Examples 12.45 and 12.46 show. Computing $nchase_F(PAD(d))$ for every update to the database is prohibitive. If the FDs of F are embodied in keys of the database scheme, there is a more efficient way to ensure d has an F-WI.

Definition 12.14 Let d be a database over the scheme $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$. Let $\mathbf{K}_i$ be a set of keys for R_i, $1 \leq i \leq p$. Let F be a set of FDs that is completely characterized by the keys of $\mathbf{R}$. Database d satisfies the *modified foreign key constraint** (MFKC) relative to F if for every $R_i \in \mathbf{R}$, if $t \in \bar{r}(R_i)$, there is a tuple t' over R_i^+ such that for any $R_j \subseteq R_i^+$, $t'(R_j) \in \bar{r}(R_j)$, and, in particular $t'(R_i) = t$.

Example 12.49 Consider the database d over scheme $\mathbf{R} = \{\underline{A}\ B, \underline{B}\ D, \underline{A}\ C, \underline{C}\ D\}$ shown in Figure 12.41. The single key for each scheme is underlined. The keys completely characterize the set of FDs $F = \{A \rightarrow B, B \rightarrow D, A \rightarrow C, C \rightarrow D\}$. Database d does not satisfy the MFKC for F. Consider $\bar{r}(A\ B)$ and the tuple $t = \langle 7\ 8 \rangle$. We see $(A\ B)^+ = A\ B\ C\ D$, but there is no tuple t' over $A\ B\ C\ D$ such that $t'(A\ B) = t$ and $t'(R) \in \bar{r}(R)$ for every other scheme $R \subseteq A\ B\ C\ D$. If we add $\langle 7\ 5 \rangle$ to $\bar{r}(A\ C)$, such a tuple exists, namely $t' = \langle 7\ 8\ 5\ 6 \rangle$. With the addition of $\langle 7\ 5 \rangle$, d does satisfy the MFKC relative to F.

$\bar{r}(A$	$B)$
1	2
7	8

$\bar{r}(B$	$C)$
2	3
5	3
8	6

$\bar{r}(A$	$C)$
1	4

$\bar{r}(C$	$D)$
4	3
5	6
5	6

Figure 12.41

If every relation in a database d satisfies its keys, and d satisfies the MFKC relative to a completely characterized set of dependencies F, then d globally satisfies F. The reason is the structure $nchase_F(PAD(d))$ will have. Let t be a tuple in $PAD(d)$ such that $DEF(t) = R_i$. That is, t came from relation $\bar{r}(R_i)$

*The *foreign key constraint* requires that if R_i contains a key K_j for R_j and $t_i \in \bar{r}(R_i)$, there is a tuple t_j in R_j such that $t_i(K_j) = t_j(K_j)$.

in d. If t^* is the corresponding tuple for t in $nchase_F(PAD(d))$, $DEF(t^*) \supseteq R_i^+$, by the MFKC. In fact, $t^*(R_i^+)$ will be exactly the tuple t' for t required by the MFKC. It follows $t^*(R_j) \in \bar{r}(R_j)$ for any relation scheme $R_j \subseteq R_i^+$. $DEF(t^*)$ can be no larger than R_i^+, because we are chasing with the key dependencies of d (or an equivalent set of FDs).

Suppose t^* enters into a violation of F along with another tuple u^* in $nchase_F(PAD(d))$. Assume u^* came from a tuple u in $PAD(d)$ where $DEF(u) = R_k$. Since the keys of d completely characterize F, t^* and u^* must violate $K_j \rightarrow R_j$ for some relation scheme R_j and key $K_j \in \mathbf{K}_j$. Both R_i^+ and R_k^+ contain R_j, since they are both definite on K_j. By the remarks in the last paragraph, $t^*(R_j) \in \bar{r}(R_j)$ and $u^*(R_j)$. If t^* and u^* violate $K_j \rightarrow R_j$, so does $\bar{r}(R_j)$. We have argued for the following result.

Theorem 12.2. Let d be a database scheme, and let F be a set of FDs completely characterized by the keys of d. If d satisfies the MFKC relative to F, and each relation in d satisfies its keys, then d globally satisfies F.

Example 12.50 Figure 12.42 shows $s = PAD(d)$ for the database d of Figure 12.41, with the addition of $\langle 7\ 5\rangle$ to $\bar{r}(A\ C)$. Figure 12.43 shows $s = nchase_F(PAD(d))$, where F is the set of FDs from Example 12.48. The nulls in s can be filled in to get an F-WI for d.

$PAD(d)$(A	B	C	D)
1	2	$\perp_1$	$\perp_2$
7	8	$\perp_3$	$\perp_4$
$\perp_5$	2	$\perp_6$	3
$\perp_7$	5	$\perp_8$	3
$\perp_9$	8	$\perp_{10}$	6
1	$\perp_{11}$	4	$\perp_{12}$
7	$\perp_{13}$	5	$\perp_{14}$
$\perp_{15}$	$\perp_{16}$	4	3
$\perp_{17}$	$\perp_{18}$	5	6

Figure 12.42

12.5.6 Independence

Enforcing the MFKC still involves checking multiple relations when making updates to a single relation. We would like a way to guarantee global satisfaction that only requires checking the relation being updated.

$s(A$	B	C	$D)$
1	2	4	3
7	8	5	6
$\perp_5$	2	$\perp_6$	3
$\perp_7$	5	$\perp_8$	3
$\perp_9$	8	$\perp_{10}$	6
$\perp_{15}$	$\perp_{16}$	4	3
$\perp_{17}$	$\perp_{18}$	5	6

Figure 12.43

Definition 12.15 Let **R** be a database scheme and let F be a set of FDs that applies to **R**. **R** is *independent for* F if every database d over **R** that obeys F (each relation satisfies the applicable FDs) globally satisfies F.

Independence can be paraphrased as "local satisfaction guarantees global satisfaction." In the case where F is embodied by the keys of **R**, there is a necessary and sufficient condition for independence. For the remainder of this section we assume that we have a relation scheme $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$, where each R_i has a set of keys $\mathbf{K}_i$. Let $F_i = \{K \rightarrow R_i | K \in \mathbf{K}_i\}$, $1 \leq i \leq p$. Let

$$F = \bigcup_{i=1}^{p} F_i \text{ and } F_{-j} = \bigcup_{\substack{i=1 \\ i \neq j}}^{p} F_i.$$

X^+ will denote the closure of a set of attributes relative to F, while X^+_{-j} denotes the closure relative to F_{-j}.

Definition 12.16 Let **R** and F be as given above. Relation scheme $R_i \in \mathbf{R}$ satisfies the *uniqueness condition* relative to F if there is no $R_j \in \mathbf{R}$, $i \neq j$, such that for some key $K \in \mathbf{K}_j$ and some attribute $B \in R_j - K$,

$$K\,B \in (R_i)^+_{-j}.$$

R satisfies the *uniqueness condition* relative to F if every $R_i \in \mathbf{R}$ satisfies it.

The uniqueness condition captures the idea that for any set of attributes X, there is exactly one way to compute X^+. That is, for any two F-based DDAGs H_1 and H_2 for $X \rightarrow X^+$, if attribute B was added to H_1 using an FD in F_i, then some FD in F_i was used to add B to H_2 (see Exercise 12.57). If **R** satisfies the uniqueness condition relative to F, then **R** is in BCNF under F (see Exercise 12.58). In particular, no dependency $K \rightarrow A$ in F may apply to more than one scheme in **R**.

Example 12.51 Let $\mathbf{R} = \{R_1, R_2, R_3, R_4\}$ for $R_1 = \underline{A}\ B$, $R_2 = \underline{B}\ D$, $R_3 = \underline{A}\ C$ and $R_4 = \underline{C}\ D$, where each scheme has the single key underlined. These keys embody the set of dependencies $F = \{A \rightarrow B, B \rightarrow D, A \rightarrow C, C \rightarrow D\}$, with extraneous attributes removed. **R** does not satisfy the uniqueness condition relative to F. Consider R_1 and R_4. $F_{-4} = \{A \rightarrow B, B \rightarrow D, A \rightarrow C\}$. Thus $(R_i)^{\pm}_4 = A\ B\ C\ D$. C is a key of R_4, and $D \in R_4 - B$. $B\ D \subseteq (R_1)^{\pm}_4$, so R_1 violates the uniqueness assumption.

Example 12.52 Let $\mathbf{R} = \{R_1, R_2, R_3\}$ for $R_1 = \underline{A\ B}\ C$, $R_2 = \underline{B\ C}\ D$ and $R_3 = \underline{A\ D}\ E$, where each scheme has the single key underlined. $F = \{A\ B \rightarrow C, B\ C \rightarrow D, A\ D \rightarrow E\}$. **R** satisfies the uniqueness assumption relative to F. For example, $(R_1)^{\pm}_2 = R_1$, which contains $B\ C$ but not D from R_2. Also, $(R_1)^{\pm}_3 = A\ B\ C\ D$, which contains $A\ D$ but not E from R_3.

Before presenting the main theorem of this section, we make two observations.

Observation 1 Let d be a database over **U** and let F be a set of FDs over **U**. Suppose we form d' by adding tuples to some or all of the relations in d. If d' has any F-WI s, then s is an F-WI for d.

Definition 12.17 Let $s \in Rel\uparrow(R)$. Let t and u be tuples in s. We say t *supersedes* u in s if $t_1(A) = t_2(A)$ whenever $t_2(A)$ is a value, or a marked null that appears elsewhere in s.

Observation 2 Let $s \in Rel\uparrow(R)$ and let F be a set of FDs over R. Let t be a tuple in s that supersedes another tuple u in s. If t^* and u^* are the tuples in $s^* = nchase_F(s)$ corresponding to t and u in s, then t^* supersedes u^* in s^*.

From Observation 2 we conclude that if we are computing $nchase_F$ $(PAD(d))$ to determine if d has an F-WI, we can delete a superseded tuple at any time, without changing the determination.

Theorem 12.3 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ be a database scheme over **U** and let $\mathbf{K}_i$, $1 \leq i \leq p$, F_i, $1 \leq i \leq p$, and F be as defined previously. **R** is independent for F if and only if **R** satisfies the uniqueness condition relative to F.

Proof (if) We assume $\mathbf{R}$ satisfies the uniqueness condition relative to F. We show that any database d over R that satisfies the keys of $\mathbf{R}$ has an F-WI. We exhibit a database d' that has an F-WI and is an extension of d. By Observation 1, we conclude d has an F-WI. We shall compute $nchase_F(PAD(d'))$ and show that we never encounter a hard violation. We actually start off computing $nchase_F(PAD(d))$. Along the way we add tuples to d and hence to $PAD(d)$. Adding tuples in the middle of the *nchase* computation will not cause problems, for we can pretend the tuples were present from the beginning and we simply did not touch them before a certain point.

We shall organize the computation of the *nchase* in such a way that if t^* is a tuple in $PAD(d)$ at some intermediate step, $DEF(t^*)$ will be the union of schemes from $\mathbf{R}$. Further, t^* will never have any nulls that appear elsewhere in $PAD(d)$.

Let $s^* \in Rel\uparrow(\mathbf{U})$ be the state of $PAD(d)$ at some point in the *nchase* computation. Suppose we are about to apply an FD $K \rightarrow A$ to tuples t^* and u^* in s^* that agree on K. Assume, inductively, that $t^*(K)\downarrow$. By the nature of F, K is a key for some scheme R_i, and $A \in R_i - K$. First, instead of just equating t^* and u^* on A, we equate them on all of $R_i - K$ at once (a slight extension of the fill-in rule). Second, we want to avoid equating marked nulls between t^* and u^*. If there is a tuple $v \in \overline{r}(R_i)$ such that $v(K) = t^*(K) = u^*(K)$, we have no problem. There must be a tuple v^* in s^* such that $v^*(K) = t^*(K) = u^*(K)$ and $v^*(R_i)\downarrow$. Instead of equating nulls between t^* and u^*, we change $t^*(R_i - K)$ and $u^*(R_i - K)$ to match $v^*(R_i - K)$. (Equating t^*, u^* and v^* on $R_i - K$ actually takes two steps in the *nchase* computation.)

Unfortunately, we have no guarantee that $\overline{r}(R_i)$ contains a tuple v with $v(K) = t^*(K) = u^*(K)$. At this point we must add such a tuple v to $\overline{r}(R_i)$. If we let $v(R_i - K)$ be values not already appearing in $\overline{r}(R_i)$, no keys for R_i will be violated. (Why?) We then pad v using new marked nulls to be a partial tuple over $\mathbf{U}$, call it v^*. We add v^* to s^* and proceed to change $t^*(R_i - K)$ and $u^*(R_i - K)$ to match $v^*(R_i - K)$. Watch carefully, please. After the changes to t^*, $t^*(R_i) = v^*(R_i)$, and v^* is unmatched marked nulls on $\mathbf{U} - \mathbf{R}_i$. By Observation 2, we may remove v^* from s^*, since t^* supersedes it. We remove v^* to ensure the *nchase* computation eventually halts.

The effect of this maneuvering is tantamount to promoting $t^*(B)$ to a value whenever $t^*(B)$ and $u^*(B)$ are marked nulls, for $B \in R_i - K$. We assume for the rest of the proof that if s^* is some state in the computation of $nchase_F(PAD(d))$, and t^* is a tuple in s^*, then $DEF(t^*)$ is the union of schemes in $\mathbf{R}$. Furthermore, we assume t^* contains no matched nulls.

As a consequence of these assumptions about computing $nchase_F(PAD(d))$, we can guarantee certain other conditions. Let t be a tuple in $PAD(d)$ such that $DEF(t) = R_i$. Let t^* be the corresponding tuple in some state s^* of the computation of $nchase_F(PAD(d))$. For $R_j \in \mathbf{R}$, if $R_j \not\subseteq DEF(t^*)$, then $(R_i)^{\pm}_j \supseteq DEF(t^*)$. No dependency arising from keys in R_j has been used to fill-in t^*, or else $DEF(t^*) \supseteq R_j$. It follows that if $DEF(t^*)$ contains a key K of R_j and an attribute $A \in R_j - K$, then $DEF(t^*)$ contains all of R_j. Otherwise, $(R_i)^{\pm}_j$ contains KA, and the uniqueness condition is violated.

After all these machinations, we must still show that we never encounter a hard violation when computing $nchase_F(PAD(d))$. Initially, there is no hard violation in $PAD(d)$, since all the relations in d obey their keys and no two relation schemes can embody a common dependency. Tuples added to $PAD(d)$ along the way are chosen so as not to violate keys. Suppose at some state in the computation of $nchase_F(PAD(d))$ we first encounter a hard violation. Let s^* be the state, and let t^* and u^* violate the FD $K \rightarrow A$. Recall that t^* and u^* must both be definite on KA. Let $t^*(A) = a$ and $u^*(A) = a'$. Since the hard violation first appeared at state s^*, one of t^* or u^* must have just been changed. Assume t^* was changed, and roll the computation back one step, to before t^* was changed. We consider three cases, which depend upon which parts of t^* are about to be filled in.

Case 1 An FD $Y \rightarrow A$ was used to fill in $t^*(A)$ as a using a tuple v^*. $K \neq Y$, otherwise, there is already a hard violation of $K \rightarrow A$ between u^* and v^*. K and Y are not in the same $\mathbf{K}_j$. If they were, then $R_j \supseteq KYA$. $DEF(t^*)$ contains K and Y, but not A, hence not R_j. We noted that such a situation does not occur, from the way we are computing the *nchase*. Let t be the tuple in $PAD(d)$ from which t^* came. Let $DEF(t) = R_i$. Assume $Y \rightarrow A$ comes from R_j. Before $t^*(A)$ is filled in, $DEF(t^*) \not\supseteq R_j$, so $(R_i)^{\pm}_j \supseteq KY$. $K \rightarrow A$ does not come from R_j, so $(R_i)^{\pm}_j \supseteq KYA$, which means R_i violates the uniqueness condition with respect to R_j.

Case 2 An FD $Y \rightarrow K'$ was used to fill in $t^*(K')$ from a tuple v^*, where $K = K'W$ for some set W (possibly empty). K and Y are not from the same $\mathbf{K}_j$, since t^* is definite on Y and W, but not K', hence not on R_j. (If $W = \emptyset$, $A \notin Y$, or else u^* and v^* violate $K \rightarrow A$. If $A \notin Y$, we have $DEF(t^*)$ containing YA but not $K' = K$, hence not R_j.) Assume t^* came from t in $PAD(d)$ where $DEF(t) = R_i$. Assume $K \rightarrow A$ comes from R_j. Before t^* is filled in on K', $DEF(t^*) \not\supseteq R_j$, so $(R_i)^{\pm}_j \supseteq YWA$. Since $Y \rightarrow K'$ is not from R_j, $(R_i)^{\pm}_j \supseteq YK'WA = YKA$. Thus, R_i violates the uniqueness condition relative to R_j.

Case 3 $Y \rightarrow K'A$ was used to fill in $t^*(K'A)$ from v^*, where $K = K'W$ for some W. $W \neq \emptyset$, or else u^* and v^* violate $K \rightarrow A$. K and Y are not from the same R_j, since $DEF(t^*)$ contains YW but not K', hence not R_j. Let t^* come from t in $PAD(d)$ where $DEF(t) = R_i$. Assume $K \rightarrow A$ is from R_j. Before t^* is filled in on $K'A$, $DEF(t^*) \not\supseteq R_j$, so $(R_i)^{\pm}_j \supseteq YW$. $Y \rightarrow K'A$ is not from R_j, so $(R_i)^{\pm}_j \supseteq YK'WA = YKA$. Hence, R_i violates the uniqueness condition relative to R_j.

In every case, we get a contradiction to the uniqueness assumption. We conclude the computation of $nchase_F(PAD(d'))$ never encounters a hard violation of F, where d' is d with some tuples added. Database d', and hence database d, has an F-WI. **R** is independent relative to F.

(only if) This part of the proof is simpler and is left to the reader (see Exercise 12.59).

Returning to window functions, consider $[\cdot]_F$ where F is the set of FDs embodied by the keys of the database scheme, **R**. Assume **R** satisfies the uniqueness condition. There will be no problems computing $[X,d]_F$ if d locally satisfies the keys of R, since an F-WI is guaranteed to exist. In that case, $[X,d]_F$ can be computed efficiently with joins (see Exercise 12.60). Also, $[\cdot]_F$ is faithful, if every relation has a non-trivial key (see Exercise 12.61).

12.5.7 A Further Condition on Window Functions

In this final section, we consider a further condition on window functions, which some will, and some will not, construe as a consequence of URSA. The faithfulness condition was imposed to insure agreement between stored relations and the window function. Still, the faithfulness condition does not totally guarantee the integrity of the semantics of stored relations. A window function can be faithful, yet induce connections on subsets of a relation scheme that are not part of the meaning of the scheme.

Example 12.53 Consider the database scheme **R** = {$\underline{\text{PLANE\# CITY}}$ HANGAR#, $\underline{\text{CITY HANGAR\#}}$ MECHANIC, $\underline{\text{PLANE\# MECHANIC}}$ LASTSERV}, where the only keys are those underlined. **R** is the same scheme as in Example 12.52, up to renaming. Let F be the embodied FDs for **R**. The intended meaning of a tuple $\langle p\ c\ h\rangle$ over PLANE# CITY HANGAR# is that plane p goes to hangar h for servicing when in city c. A tuple $\langle c\ h\ m\rangle$ over CITY HANGAR# MECHANIC means that m is in charge of service for hangar h in city c. A tuple $\langle p\ m\ l\rangle$ over PLANE# MECHANIC LASTSERV means mechanic m last serviced plane p on date l. The window function $[\cdot]_F$ is the same as the window function $[\cdot]_{\mathbf{R},\mathbf{O}}$, where $\mathbf{O} = \mathbf{R} \cup$ {PLANE# CITY HANGAR# MECHANIC, PLANE# CITY HANGAR# MECHANIC LASTSERV}. **R** and **O** are diagrammed in Figure 12.52.

Consider a tuple $\langle p\ m \rangle$ from $[\text{PLANE\# MECHANIC}]_F$. If we just consider the meaning of PLANE# MECHANIC LASTSERV, we would conclude that $\langle p\ m \rangle$ means there is some date l such that m serviced p on l. However, it could result from tuples $\langle p\ c\ h \rangle$ and $\langle c\ h\ m \rangle$ in $\bar{r}(\text{PLANE\# CITY HANGAR\#})$ and $\bar{r}(\text{CITY HANGAR\# MECHANIC})$. That is, if p were serviced in city c, it would be by m. This second meaning does not follow from PLANE# MECHANIC LASTSERV, which implies m already worked on p. The difference is potentiality versus actuality. Somehow, $[\cdot]_F$ does not preserve the integrity of PLANE# MECHANIC LASTSERV.

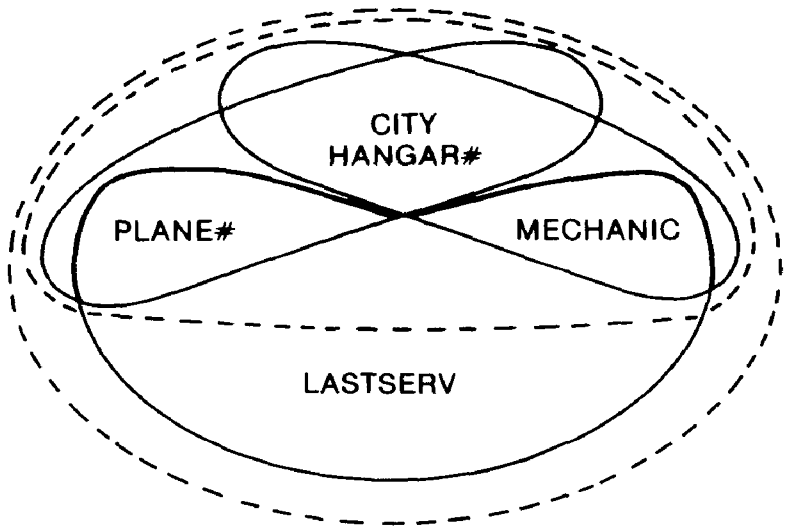

Figure 12.44

Definition 12.18 Let **R** be a database scheme and let $[\cdot]$ be a window function for databases over **R**. The *strong faithfulness condition* on $[\cdot]$ requires that for any scheme $R \in \mathbf{R}$, any $X \subseteq R$, and any database d over **R**, $[X, d]$ depends only on the states of relations whose schemes are contained in R, that is

$$\{\bar{r}(S) \mid S \in \mathbf{R},\ S \subseteq R\}.$$

Strong faithfulness is violated in Example 12.53, since $[\text{PLANE\# MECHANIC}]_F$ does not depend solely on $\bar{r}(\text{PLANE\# MECHANIC LASTSERV})$. Recall from Example 12.52 that **R** does satisfy the uniqueness condition, so the uniqueness condition does not assure a window function is strongly faithful (see Exercise 12.70). For window functions of the type $[\cdot]_{\mathbf{R},\mathbf{O}}$, we can give a sufficient condition for strong faithfulness.

Theorem 12.4 Let $\mathbf{R}$ be a database scheme over $\mathbf{U}$ and let $\mathbf{O}$ be a set of objects containing R. Assume every object in $\mathbf{O}$ is the union of schemes in $\mathbf{R}$. The window function $[\cdot]_{\mathbf{R},\mathbf{O}}$ is strongly faithful if $\mathbf{O}$ is closed under nonempty intersection. (That is, for $W, Z \in \mathbf{O}$, if $W \cap Z \neq \emptyset$ then $W \cap Z \in \mathbf{O}$.)

Proof We first show that for any subset X of $\mathbf{U}$ and database d, $[X,d]_{\mathbf{R},\mathbf{O}}$ can be computed from $[Y,d]_{\mathbf{R},\mathbf{O}}$ for a single $Y \in \mathbf{O}$, provided $\mathbf{O}$ is closed under intersection. Recall

$$[X]_{\mathbf{R},\mathbf{O}} = \bigcup_{\substack{Y \in \mathbf{O} \\ Y \supseteq X}} \pi_X([Y]_{\mathbf{R},\mathbf{O}}).$$

Let Y_1 and Y_2 both be in $\mathbf{O}$ and both contain X. $Y_1 \cap Y_2 = Y_3$ is nonempty, hence in $\mathbf{O}$. By the containment condition on the window function, $\pi_X([Y_3]_{\mathbf{R},\mathbf{O}})$ contains both $\pi_X([Y_1]_{\mathbf{R},\mathbf{O}})$ and $\pi_X([Y_2]_{\mathbf{R},\mathbf{O}}$. Hence Y_1 and Y_2 can be dropped from the union for computing $[X]_{\mathbf{R},\mathbf{O}}$ without changing the result. Proceeding in this manner, we may remove all but one object from the union, call it Y. We then have

$$[X]_{\mathbf{R},\mathbf{O}} = \pi_X([Y]_{\mathbf{R},\mathbf{O}}).$$

What we have demonstrated is that if X is contained in some object of $\mathbf{O}$, there is a unique minimum object Y that contains X. That is, for any W in $\mathbf{O}$ that contains X, $W \supseteq Y$. Further, $[X]_{\mathbf{R},\mathbf{O}}$ takes its value from $[Y]_{\mathbf{R},\mathbf{O}}$. Thus, if R is a scheme in $\mathbf{R}$ and $X \subseteq R$, there is a unique minimum Y in $\mathbf{O}$ with $X \subseteq Y \subseteq R$ (since $R \in \mathbf{O}$). The window function on X depends on the window function on Y, which in turn depends on

$$\{\bar{r}(S) \mid S \in \mathbf{R},\ S \subseteq Y\},$$

a subset of

$$\{\bar{r}(S) \mid S \in \mathbf{R},\ S \subseteq R\}.$$

Closure under intersection is not necessary for strong faithfulness.

Example 12.54 If $\mathbf{R} = \{B, C, AB, BD, AC, CD\}$ and $\mathbf{O} = \mathbf{R} \cup \{ABC, BCD\}$, then $[\cdot]_{\mathbf{R},\mathbf{O}}$ is strongly faithful but $\mathbf{O}$ is not closed under intersection. $ABC \cap BCD = BC$, which is not in $\mathbf{O}$.

There is a condition on window functions of the type $[\cdot]_{\mathbf{R},\mathbf{O}}$ that is equivalent to $\mathbf{O}$ being closed under intersection. The definition does not apply to window functions that are not defined directly from objects.

Definition 12.19 Let $\mathbf{R}$ be a database scheme and let $\mathbf{O}$ be a set of objects containing $\mathbf{R}$. The window function $[\cdot]_{\mathbf{R},\mathbf{O}}$ is *faithful for objects* if for any object $W \in \mathbf{O}$ and any $X \subseteq W$, $[X]_{\mathbf{R},\mathbf{O}}$ depends only on $\{\overline{r}(R) \mid R \in \mathbf{R}, R \subseteq W\}$.

Object-faithfulness prevents a little knowledge from being a dangerous thing. If $[\cdot]_{\mathbf{R},\mathbf{O}}$ is faithful for objects, knowing the meaning of $\overline{r}(R)$ for every scheme R contained in an object W provides the meaning for $[X]_{\mathbf{R},\mathbf{O}}$ for any $X \subseteq W$. Such is not the case when $[\cdot]_{\mathbf{R},\mathbf{O}}$ is not faithful for objects, as we saw in the last example. (Object-faithfulness implies strong faithfulness.)

Theorem 12.5 Let $\mathbf{R}$ be a database scheme over $\mathbf{U}$ and let $\mathbf{O}$ be a set of objects containing $\mathbf{R}$. Assume every object in $\mathbf{O}$ is the union of schemes in $\mathbf{R}$. The window function $[\cdot]_{\mathbf{R},\mathbf{O}}$ is faithful for objects if and only if $\mathbf{O}$ is closed under nonempty intersection.

Proof The "if" part follows from Theorem 12.4. The "only if" portion is left as Exercise 12.65. The basic idea is that if X is the intersection of objects Z and W, but not itself an object, then $[X]_{\mathbf{R},\mathbf{O}}$ depends on both relations from W and relations from Z.

There are generally several ways to modify a database scheme and a set of objects to make objects closed under intersection.

Example 12.55 Figures 12.45 and 12.46 show two ways to modify $\mathbf{R}$ and $\mathbf{O}$ from Example 12.53 to get closure of $\mathbf{O}$ under intersection. Under the first modification, PLANE#-MECHANIC pairs can exist independently. Under the second modification, PLANE#-MECHANIC pairs mean only that the mechanic could potentially work on the plane. A mechanic cannot have serviced a plane unless he or she could potentially have worked on it.

There are reasons for preferring closure of objects under intersection other than object faithfulness. There is a computational advantage, as shown in the proof of Theorem 12.4. A semantic argument can also be made for closure under intersection. Let W and Z be objects, with $X = W \cap Z \neq \emptyset$.

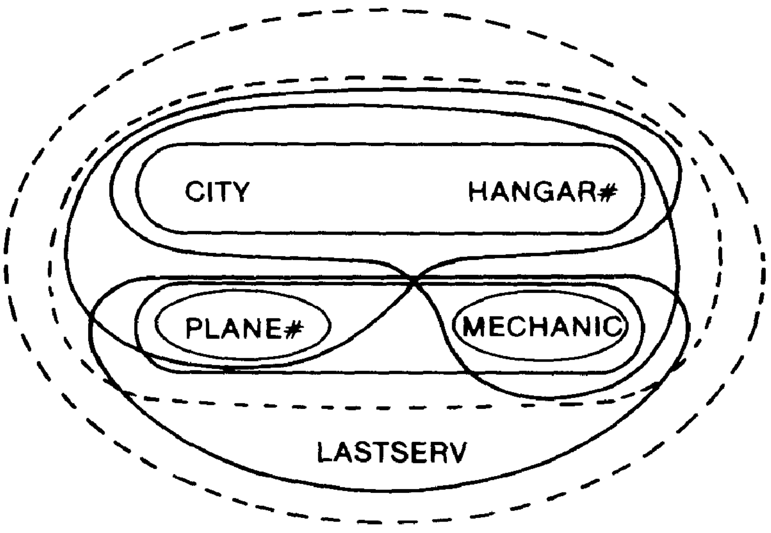

Figure 12.45

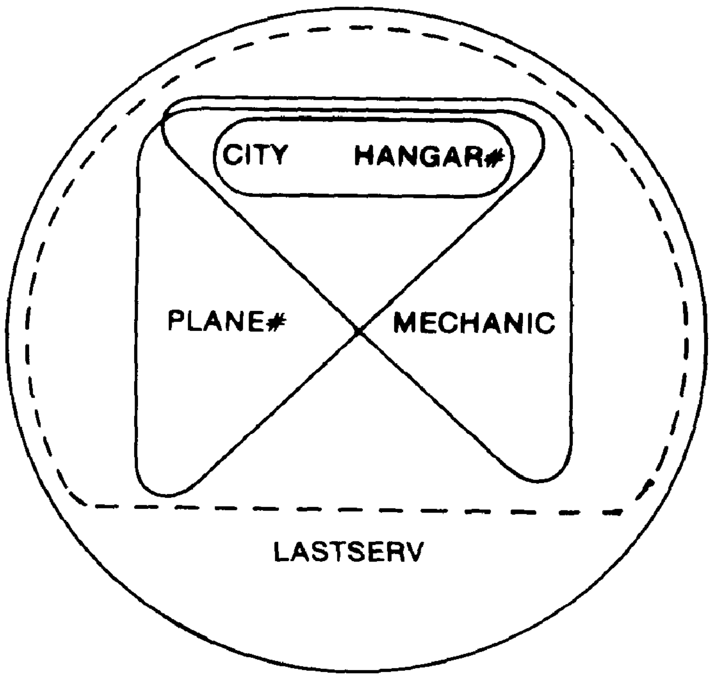

Figure 12.46

We know X-values make sense without $(W - X)$-values (in Z) and X-values make sense without $(Z - X)$-values (in W). It seems X-values should make sense with neither $(W - X)$-values nor $(Z - X)$-values: X should be an object in its own right.

Example 12.56 Referring to Example 12.53, consider CITY and HANGAR#. We can connect a CITY-HANGAR# pair without a PLANE# in $\bar{r}$(CITY HANGAR# MECHANIC). We can connect a CITY-HANGAR# pair without a MECHANIC in $\bar{r}$(PLANE# CITY HANGAR#). It would seem that a CITY-HANGAR# pair could exist without PLANE# or MECHANIC, but there is no place to store a pair. There is no place to assert that city c has a hangar h. Both the modified schemes of Example 12.55 have CITY HANGAR# as a relation scheme, and hence can store such pairs.

12.6 EXERCISES

12.1 True or False?

(a) For relations r and s, if $r \geq s$ and r and s have the same number of tuples, then $r > s$.

(b) For relations r and s, if $r \geq s$ and for every tuple $t_r \in r$ there is a tuple $t_s \in s$ such that $t_r \geq t_s$, then $r > s$.

12.2 Find a completion of the relation r below with a minimum number of tuples

$r(A$	B	C	$D)$
1	$\perp$	$\perp$	6
1	3	$\perp$	$\perp$
$\perp$	3	$\perp$	7
$\perp$	$\perp$	5	7
$\perp$	4	5	$\perp$
2	4	$\perp$	$\perp$

12.3* For a relation r with n columns and k tuples, what is the fewest tuples that any completion of r may have?

12.4 Show that $r > s$ and $s > r$ imply $r = s$.

12.5 Give relations r and s such that $r > s$ and there is a completion of s that is not a completion of r.

12.6 Let r and s be relations over scheme R. Show that if every completion of r is a completion of s, then $r > s$.

12.7 Let f be a tuple calculus formula with a single free variable, x, and that mentions a single relation, r. Give a method for evaluating the expression $\{x(R) | f(x)\}$ when r is partial. Does your method allow the value of the expression itself to be a partial relation? You might want the value to include a "sure" and a "maybe" component.

12.8 Give a tuple calculus formula involving a partial relation r such that $I_s(f)$ is *false* for every completion s of r, but $I_q(f) = true$ for some extension q of r.

12.9 Show that in the definition of permissible in Section 12.2, if "extension" is used instead of "completion," an equivalent definition results.

12.10 Give a set of FDs F and a relation r such that a particular null must take the same value in any permissible completion of r, but this value cannot be inferred using the fill-in rule for unmarked nulls. Hint: Think about nondistinguished variables in a tableau.

12.11 Characterize when two partial relations with marked nulls have the same set of completions.

12.12 Give a definition of *subsumes* for relations with marked nulls. We want a syntactic condition that characterizes when one relation, treated as a set of axioms, logically implies another.

12.13 Show that if in the definition of *augments* for relations with marked nulls, only values may replace nulls, an equivalent definition of *completion* results.

12.14 Let $r \in Rel\uparrow(R)$ and let F be a set of FDs over R. Show that arbitrary completions of $nchase_F(r)$ need not be permissible.

12.15 Let $r \in Rel\uparrow(R)$ and let F be a set of FDs over R. Let $r^* = nchase_F(r)$. Assume $r^* \neq HV$ and suppose t is a tuple in r^* with $t(A) = \perp_i$, $A \in R$. Show that r^* has permissible completions under F that do not agree on $t(A)$.

12.16 Give a correct and complete set of inference axioms for existence constraints.

12.17 Let E be a set of ECs over relation scheme R. For a subset X of R, let X^* denote the maximal set of attributes Y such that $X \urcorner Y$ is implied by E. (X^* is analogous to X^+ for FDs.) Let F be a set of FDs over R. When adding a tuple t to a relation $r \in Rel\uparrow(R)$, it may be that t initially satisfies E, but filling in nulls using F after the addition causes t to violate E. Alternatively, t may violate E initially, but filling in nulls after the addition makes t satisfy E. Using X^+, X^* and $DEF(t)$, characterize when

(a) t is guaranteed to satisfy E after insertion into r,

(b) t will possibly satisfy E after insertion, and

(c) t will surely violate E after insertion.

12.18 Show that if q is a minimal extension of r, then q is a close extension of r.

12.19 Is a minimal extension of a relation r necessarily a completion of r? Is a completion necessarily a minimal extension?

12.20 Which of the following possibility functions are reasonable?
(a) $POSS_1(r) = \{s | s\downarrow \geq r \text{ and } |s| = |r|\}$.
(b) $POSS_2(r) = \{s | s\downarrow \geq r \text{ and } |s| \leq |r|\}$.
(c) $POSS_3(r) =$
$\{s | s\downarrow \geq r$ and no proper subrelation of s completes $r\}$.
(d) $POSS_4(r) =$
$\{s | s\downarrow \geq r$ and no relation with fewer tuples than s extends $r\}$.

12.21 Show that for any closed possibility function $POSS$, the collection of sets $\{POSS(r) | r \in Rel\uparrow(R)\}$ is not closed under intersection.

12.22 Show that the collection of sets $\{POSS_O(r)\ r \in Rel\uparrow(R)\}$ is closed under intersection.

12.23 Prove that no precise generalization for join exists for any closed possibility function.

12.24 Show that if γ' is an adequate and restricted generalization of γ for a closed possibility function, then γ' is faithful to γ.

12.25 Show that for $POSS_O$, $r \sqsupseteq s$ if and only if $r \geq s$, and that $r \sqsupsetneq s$ if and only if $r \gneq s$.

12.26 Prove that for any relation $r \in Rel\uparrow(R)$ and any nontrivial JD $*[\mathbf{R}]$ over R, there is an extension of r that does not satisfy $*[\mathbf{R}]$.

12.27 Show that $\bowtie^O$ is restricted for $POSS_O$. Hint: If $q = r \bowtie^O s$ and there is a q' such that $POSS_O(q) \supseteq POSS_O(q') \supseteq POSS_O(r) \bowtie POSS_O(s)$, then $q' \sqsupseteq q$ and hence $q' \geq q$. Consider a tuple t in q' that is not subsumed by any tuple in q.

12.28 Give a precise generalization of project for $POSS_O$.

12.29 Show that $\sigma^O_{A=a}$ and $\sigma^O_{A=B}$ are adequate and restricted for $POSS_O$.

12.30 Prove that $POSS_{CE}$ and $POSS_{ME}$ are reasonable possibility functions.

12.31 (a) Give adequate and restricted generalizations of union and project for $POSS_{CE}$.
(b) Show that there is no adequate and restricted generalization of join for $POSS_{CE}$.

12.32 Do adequate and restricted generalizations for union, join, and select exist for $POSS_C$ and $POSS_{ME}$?

12.33* Give syntactic characterizations for $r \sqsupseteq s$ for the possibility functions $POSS_{CE}$, $POSS_C$ and $POSS_{ME}$.

12.34 Are there adequate and restricted generalizations for intersection, complement, and select with inequality comparisons for any of $POSS_O$, $POSS_{CE}$, $POSS_C$, and $POSS_{ME}$?

12.35 Show that if an operator γ has an adequate and restricted generalization γ' for possibility function, and γ' is not precise, then no precise generalization of γ exists.

12.36 Give rules for redundant tuple removal for $POSS_{CE}$, $POSS_C$ and $POSS_{ME}$.

12.37 Prove the corollary to Lemma 12.2.

12.38 Prove the converse of Lemma 12.3.

12.39 Show that the definition of π_X^B is adequate and restricted for $POSS_B$. Is it precise?

12.40 Give an adequate and restricted generalization $\sigma_{A=B}^B$ of $\sigma_{A=B}$ for $POSS_B$. What can be deduced about $\sigma_{A=B}^B(\sigma_{A=a}^B(r))$ versus $\sigma_{A=a}^B(\sigma_{A=B}^B(r))$?

12.41 Show that $\cup^B$ is precise for $POSS_B$.

12.42 Prove the claim in the proof of Lemma 12.5.

12.43 Which of the generalized operators defined for $POSS_O$ and $POSS_B$ are faithful? Which of the generalized binary operators are associative and commutative?

12.44* Let **O** be a set of objects over a set of attributes **U** and suppose **U** is one of the objects in **O**. Show that there exists a set E of ECs such that

$$\{DEF(t) \mid t \text{ satisfies } E\} = \mathbf{O}$$

12.45 Show that the extension of a window function from objects to all sets of attributes given in Section 12.5 preserves the containment condition and gives the same value as the original function on objects.

Definition 12.20 Given a set of objects **O** and a set of attributes X, $W \in \mathbf{O}$ is a *minimal object for* X if $W \supseteq X$ and there is no object Z in **O** such that $W \supsetneq Z \supseteq X$.

12.46 Prove that the value of an extended window function on a set of attributes X can be computed from the value of the original window function on the minimal objects for X.

12.47 Show that the window function defined in Section 12.5.4 satisfies the containment condition, even if the underlying database does not.

Definition 12.21 Given a database scheme **R** and a set of attributes X, $R \in \mathbf{R}$ is a *maximal scheme for* X if $X \supseteq R$ and there is no scheme S in **R** such that $X \supseteq S \supsetneq R$.

12.48 Using the window function from Section 12.5.4, and assuming the underlying database satisfies the containment condition, show that $[X,d]_{\mathbf{R},\mathbf{O}}$ is the join of every relation $^{-}r(R)$ where R is a maximal scheme for object $X \in \mathbf{O}$.

12.49 Does a database d necessarily have a representative instance s such that for at least one relation $r(R)$ in d, $\pi_R(s) = r$?

12.50 Prove Theorem 12.1. You will need to make some assumptions to handle the case where $nchase_{\mathbf{C}}(PAD(d)) = HV$.

12.51 Show that the window function $[\cdot]_{*\mathbf{R}}$ is faithful on any database satisfying the containment condition.

12.52* What are the implicit objects for $[\cdot]_{*\mathbf{R}}$ for $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$? Define $[W]_{*\mathbf{R}}$ for implicit object W in terms of joins.

12.53 Let d be a database with scheme $\mathbf{R}$ over $\mathbf{U}$ and let F be a set of FDs over $\mathbf{U}$. Let t be a tuple in $PAD(d)$. Let t^* be the corresponding tuple in $nchase_F(PAD(d))$. Prove $DEF(t)^+ \supseteq DEF(t)$.

12.54 Show that the foreign key constraint is not sufficient to guarantee that a locally satisfying database globally satisfies a set of FDs.

12.55 Show that $[\cdot]_F$ is faithful for a database d that satisfies the MFKC relative to F.

12.56* Consider $[\cdot]_F$ on databases over scheme $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$ that satisfy the MFKC relative to F. Give the implicit objects for $[\cdot]_F$, and for each object W, give a definition of $[W]_F$ using joins.

12.57 Let $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$, $F_1, F_2, \ldots, F_p$, and F be defined as in Section 12.5.6. Let $\mathbf{U} = R_1R_2 \cdots R_p$ and let $X \subseteq \mathbf{U}$. Assuming $\mathbf{R}$ satisfies the uniqueness condition, show that the construction of any F-based DDAG for $X \rightarrow X^+$ is unique relative to $F_1, F_2, \ldots, F_p$. For any attribute $A \in X^+ - X$, there is a unique j, depending on A and X, such that a node for A can only be added to the DDAG by using an FD from F_j.

12.58 Prove that the uniqueness condition implies BCNF.

12.59 Prove the "only if" portion of Theorem 12.3. If $\mathbf{R}$ violates the uniqueness condition relative to F, exhibit a database d that locally satisfies F but does not globally satisfy F. Hint: Such a database exists where every relation has but a single tuple.

12.60* Let $\mathbf{R}$ be a database scheme and let F be the set of FDs embodied in the keys of $\mathbf{R}$. For a database d on $\mathbf{R}$ and a set of attributes X, show that $[X,d]_F$ can be computed using union, projection, and extension joins.

12.61* The uniqueness condition does not allow database schemes where one relation scheme is the subscheme of another, unless the subscheme has a single, trivial key. Suppose we modify the definition of the uniqueness condition so that $(R_i)^{\pm}_{j}$ may contain a key and another attribute from R_j if $R_i \supseteq R_j$. Alter the definition of independence to be that local satisfaction *and* the containment condition imply global satisfaction. Show that Theorem 12.3 holds under the altered definitions.

12.62 Let **R** be a database scheme whose keys embody the set of FDs F. Show that $[\cdot]_F$ is faithful if **R** satisfies the uniqueness condition. Show that $[\cdot]_F$ can be faithful even if **R** violates the uniqueness condition and every relation has a non-trivial key.

12.63 Show that for any database scheme **R**, the window function $[\cdot]_{*\mathbf{R}}$ is strongly faithful.

12.64 Let **R** be a database scheme whose keys embody a set F of FDs. Find a necessary condition for $[\cdot]_F$ to be strongly faithful.

12.65 Complete the proof of Theorem 12.4. Let X be a non-object that is the intersection of objects W and Z. Show that $[X]_{\mathbf{R},\mathbf{O}}$ can be modified by changing relations whose schemes are in Z but not in W.

12.7 BIBLIOGRAPHY AND COMMENTS

Missing and partial information are problems in any database model. The ANSI/X3/SPARC report [1975] lists over a dozen types of nulls, although some are artifacts of the workings of the database system. Codd [1975] suggested the null substitution principle for evaluating expressions with partial relations. Grant [1977] pointed out that no recursive interpretation function can be defined for null substitution. Vassiliou [1979] considers algorithms for evaluating expressions under null substitution.

Walker [1979, 1980b] first suggested using dependencies to fill in nulls, although his method deals only with unmarked nulls. Marked nulls are from Maier [1980a], Vassiliou [1980a] and Honeyman [1980b, 1980c]. Lien [1979] considers the interaction of nulls and MVDs. Grant [1979] and Lipski [1979b, 1981] treat nulls that represent intervals or sets of values, rather than all possible values. Existence constraints are from Maier [1980a]. Goldstein [1981a] generalizes ECs to constraints on all types of values. Rozenshtein [1981] looks at efficient implementations of partial relations. Possibility functions follow from Biskup [1980a, 1981], who also introduces $POSS_B$ and generalized operators for $POSS_B$. Codd [1975], Lacroix and Pirotte [1976], and Zaniolo [1977] present generalized join operators. Reiter [1978] discussed the closed world assumption as it applies to databases.

Honeyman, Ladner, and Yannakakis [1980] show that testing the UIA is NP-complete in general. Other criticism of the UIA and URSA is given by Kent [1979b], Bernstein and Goodman [1980b], and Atzeni and Parker

[1981]. Objects are briefly mentioned by Zaniolo [1977] and are treated in detail by Sciore [1980b]. Carlson and Kaplan [1976] and Osborn [1979b] discuss automatically connecting relations in a database. Window functions are from Maier [1980a]. The window function $[\cdot]_{R,O}$ was developed in a series of papers by Korth and Ullman [1980], Maier and Ullman [1980], Korth [1981], and Maier and Warren [1981b]. Representative instances were developed by Honeyman [1980b, 1980c], Graham [1981a], and Mendelzon [1981]. The window function $[\cdot]_{*R}$ is from Yannakakis [1981]. The MFKC was proposed by Sagiv [1981a]. Stein [1981] considers efficiently enforcing the MFKC. The uniqueness condition and its equivalence to independence are due to Sagiv [1981b]. The proof of Theorem 12.8 used here was arrived at with the help of Ed Sciore.

Numerous semantic extensions to the relation model have been proposed, such as those by Schmid and Swenson [1975], Codd [1979], Kent [1979a], and Housel, Waddle, and Yao [1979]. Clifford and Warren [1981] consider the semantics of time in relational databases. The usefulness of summary information in databases is studied by LeViet, Kambayashi, *et al.* [1979], Walker [1980a], and Bernstein, Blaustein, and Clarke [1980].

URSA requires renaming related attributes. Bachman and Daya [1977], Smith and Smith [1977a, 1977b], Gewirtz [1979], and Sciore [1979, 1980a] all deal with renaming attributes or capturing information about related attributes, particularly where one attribute represents a specialized role of another attribute.

For more material on semantics of databases, the reader is referred to Sundgren [1975], Chen [1976], Sowa [1976], the collections edited by Nijssen [1976, 1977], Brodie [1978], and Hammer and McLeod [1978, 1980].

Exercise 12.44 is from Goldstein [1980, 1981b]. Exercise 12.53 is motivated by Yannakakis [1981]. Sagiv [1981a, 1981b] gives answers to parts of Exercises 12.56 and 12.60.