8 Functional and Join Dependency

Alice: *Your model reduces the most interesting information to something flat and boring.*

Vittorio: *You're right, and this causes a lot of problems.*

Sergio: *Designing the schema for a complex application is tough, and it is easy to make mistakes when updating a database.*

Riccardo: *Also, the system knows so little about the data that it is hard to obtain good performance.*

Alice: *Are you telling me that the model is bad?*

Vittorio: *No, wait, we are going to fix it!*

This chapter begins with an informal discussion that introduces some simple dependencies and illustrates the primary motivations for their development and study. The two following sections of the chapter are devoted to two of the simple kinds of dependencies; and the final section introduces the chase, an important tool for analyzing these dependencies and their effect on queries.

Many of the early dependencies introduced in the literature use the named (as opposed to unnamed) perspective on tuples and relations. Dependency theory was one of the main reasons for adopting this perspective in theoretical investigations. This is because dependencies concern the semantics of data, and attribute names carry more semantics than column numbers. The general view of dependencies based on logic, which is considered in Chapter 10, uses the column-number perspective, but a special subcase (called *typed*) retains the spirit of the attribute-name perspective.

8.1 Motivation

Consider the database shown in Fig. 8.1. Although the schema itself makes no restrictions on properties of data that might be stored, the intended application for the schema may involve several such restrictions. For example, we may know that there is only one director associated with each movie title, and that in *Showings*, only one movie title is associated with a given theater-screen pair.[1] Such properties are called *functional dependencies* (fd's) because the values of some attributes of a tuple uniquely or *functionally* determine the values of other attributes of that tuple. In the syntax to be developed in this chapter, the

[1] Gone are the days of seeing two movies for the price of one!

Movies

Title	*Director*	*Actor*
The Birds	Hitchcock	Hedren
The Birds	Hitchcock	Taylor
Bladerunner	Scott	Hannah
Apocalypse Now	Coppola	Brando

Showings

Theater	*Screen*	*Title*	*Snack*
Rex	1	The Birds	coffee
Rex	1	The Birds	popcorn
Rex	2	Bladerunner	coffee
Rex	2	Bladerunner	popcorn
Le Champo	1	The Birds	tea
Le Champo	1	The Birds	popcorn
Cinoche	1	The Birds	Coke
Cinoche	1	The Birds	wine
Cinoche	2	Bladerunner	Coke
Cinoche	2	Bladerunner	wine
Action Christine	1	The Birds	tea
Action Christine	1	The Birds	popcorn

Figure 8.1: Sample database illustrating simple dependencies

dependency in the *Movies* relation is written as

$$Movies : Title \rightarrow Director$$

and that of the *Showings* relation is written as

$$Showings : Theater\ Screen \rightarrow Title.$$

Technically, there are sets of attributes on the left- and right-hand sides of the arrow, but we continue with the convention of omitting set braces when understood from the context.

When there is no confusion from the context, a dependency $R : X \rightarrow Y$ is simply denoted $X \rightarrow Y$. A relation I *satisfies* a functional dependency $X \rightarrow Y$ if for each pair s, t of tuples in I,

$$\pi_X(s) = \pi_X(t) \text{ implies } \pi_Y(s) = \pi_Y(t).$$

An important notion in dependency theory is *implication*. One can observe that any relation satisfying the dependency

(a) $$Title \rightarrow Director$$

also has to satisfy the dependency

(b) $$Title, Actor \rightarrow Director.$$

We will say that dependency (a) implies dependency (b).

A *key dependency* is an fd $X \rightarrow U$, where U is the full set of attributes of the relation. It turns out that dependency (b) is equivalent to the key dependency *Title, Actor* $\rightarrow$ *Title*, *Director*, *Actor*.

A second fundamental kind of dependency is illustrated by the relation *Showings*. A tuple (*th*, *sc*, *ti*, *sn*) is in *Showings* if theater *th* is showing movie *ti* on screen *sc* and if theater *th* offers snack *sn*. Intuitively, one would expect a certain independence between the *Screen-Title* attributes, on the one hand, and the *Snack* attribute, on the other, for a given value of *Theater*. For example, because (Cinoche, 1, The Birds, Coke) and (Cinoche, 2, Bladerunner, wine) are in *Showings*, we also expect (Cinoche, 1, The Birds, wine) and (Cinoche, 2, Bladerunner, Coke) to be present. More precisely, if a relation I has this property, then

$$I = \pi_{Theater,Screen,Title}(I) \bowtie \pi_{Theater,Snack}(I).$$

This is a simple example of a *join dependency* (jd) which is formally expressed by

$$Showings : \bowtie[\{Theater, Screen, Title\}, \{Theater, Snacks\}].$$

In general, a jd may involve more than two attribute sets. *Multivalued dependency* (mvd) is the special case of jd's that have at most two attribute sets. Due to their naturalness, mvd's were introduced before jd's and have several interesting properties, which makes them worth studying on their own.

As will be seen later in this chapter, the fact that the fd *Title* $\rightarrow$ *Director* is satisfied by the *Movies* relation implies that the jd

$$\bowtie[\{Title, Director\}, \{Title, Actor\}]$$

is also satisfied. We will also study such interaction between fd's and jd's.

So far we have considered dependencies that apply to individual relations. Typically these dependencies are used in the context of a database schema, in which case one has to specify the relation concerned by each dependency. We will also consider a third fundamental kind of dependency, called *inclusion dependency* (ind) and also referred to as "referential constraint." In the example, we might expect that each title currently being shown (i.e., occurring in the *Showings* relation) is the title of a movie (i.e., also occurs in the *Movies* relation). This is denoted by

$$Showings[Title] \subseteq Movies[Title].$$

In general, ind's may involve sequences of attributes on both sides. Inclusion dependencies will be studied in depth in Chapter 9.

Data dependencies such as the ones just presented provide a formal mechanism for expressing properties expected from the stored data. If the database is known to satisfy a set of dependencies, this information can be used to (1) improve schema design, (2) protect data by preventing certain erroneous updates, and (3) improve performance. These aspects are considered in turn next.

Schema Design and Update Anomalies

The task of designing the schema in a large database application is far from being trivial, so the designer has to receive support from the system. Dependencies are used to provide information about the semantics of the application so that the system may help the user choose, among all possible schemas, the most appropriate one.

There are various ways in which a schema may not be appropriate. The relations *Movies* and *Showings* illustrate the most prominent kinds of problems associated with fd's and jd's:

Incomplete information: Suppose that one is to insert the title of a new movie and its director without knowing yet any actor of the movie. This turns out to be impossible with the foregoing schema, and it is an *insertion anomaly*. An analogue for deletion, a *deletion anomaly*, occurs if actor Marlon Brando is no longer associated with the movie "Apocalypse Now." Then the tuple ⟨Apocalypse Now, Coppola, Brando⟩ should be deleted from the database. But this has the additional effect of deleting the association between the movie "Apocalypse Now" and the director Coppola from the database, information that may still be valid.

Redundancy: The fact that Coke can be found at the Cinoche is recorded many times. Furthermore, suppose that the management of the Cinoche decided to sell Pepsi instead of Coke. It is not sufficient to modify the tuple ⟨Cinoche, 1, The Birds, Coke⟩ to ⟨Cinoche, 1, The Birds, Pepsi⟩ because this would lead to a violation of the jd. We have to modify several tuples. This is a *modification anomaly*. Insertion and deletion anomalies are also caused by redundancy.

Thus because of a bad choice for the schema, updates can lead to loss of information, inconsistency in the data, and more difficulties in writing correct updates. These problems can be prevented by choosing a more appropriate schema. In the example, the relation *Movies* should be "decomposed" into two relations *M-Director*[*Title*, *Director*] and *M-Actor*[*Title*, *Actor*], where *M-Director* satisfies the fd *Title* $\rightarrow$ *Director*. Similarly, the relation *Showings* should be replaced by two relations *ST-Showings*[*Theater*, *Screen*, *Title*] and *S-Showings*[*Theater*, *Snack*], where *ST-Showings* satisfies the fd *Theater, Screen* $\rightarrow$ *Title*. This approach to schema design is explored in Chapter 11.

Data Integrity

Data dependencies also serve as a filter on proposed updates in a natural fashion: If a database is expected to satisfy a dependency σ and a proposed update would lead to the

violation of σ, then the update is rejected. In fact, the system supports transactions. During a transaction, the database can be in an inconsistent state; but at the end of a transaction, the system checks the integrity of the database. If dependencies are violated, the whole transaction is rejected (*aborted*); otherwise it is accepted (*validated*).

Efficient Implementation and Query Optimization

It is natural to expect that knowledge of structural properties of the stored data be useful in improving the performances of a system for a particular application.

At the physical level, the satisfaction of dependencies leads to a variety of alternatives for storage and access structures. For example, satisfaction of an fd or jd implies that a relation can be physically stored in decomposed form. In addition, satisfaction of a key dependency can be used to reduce indexing space.

A particularly striking theoretical development in dependency theory provides a method for optimizing conjunctive queries in the presence of a large class of dependencies. As a simple example, consider the query

$$ans(d, a) \leftarrow Movies(t, d, a'), Movies(t, d', a),$$

which returns tuples $\langle d, a\rangle$, where actor a acted in a movie directed by d. A naive implementation of this query will require a join. Because *Movies* satisfies *Title* $\rightarrow$ *Director*, this query can be simplified to

$$ans(d, a) \leftarrow Movies(t, d, a),$$

which can be evaluated without a join. Whenever the pattern of tuples $\{\langle t, d, a'\rangle, \langle t, d', a\rangle\}$ is found in relation *Movies*, it must be the case that $d = d'$, so one may as well use just the pattern $\{\langle t, d, a\rangle\}$, yielding the simplified query. This technique for query optimization is based on the chase and is considered in the last section of this chapter.

8.2 Functional and Key Dependencies

Functional dependencies are the most prominent form of dependency, and several elegant results have been developed for them. Key dependencies are a special case of functional dependencies. These are the dependencies perhaps most universally supported by relational systems and used in database applications. Many issues in dependency theory have nice solutions in the context of functional dependencies, and these dependencies lie at the origin of the decomposition approach to schema design.

To specify a class of dependencies, one must define the syntax and the semantics of the dependencies of concern. This is done next for fd's.

DEFINITION 8.2.1 If U is a set of attributes, then a *functional dependency* (fd) over U is an expression of the form $X \rightarrow Y$, where $X, Y \subseteq U$. A *key dependency* over U is an fd of the form $X \rightarrow U$. A relation I over U *satisfies* $X \rightarrow Y$, denoted $I \models X \rightarrow Y$, if for each

pair s, t of tuples in I, $\pi_X(s) = \pi_X(t)$ implies $\pi_Y(s) = \pi_Y(t)$. For a set Σ of fd's, I *satisfies* Σ, denoted $I \models \Sigma$, if $I \models \sigma$ for each $\sigma \in \Sigma$.

A functional dependency over a database schema $\mathbf{R}$ is an expression $R : X \to Y$, where $R \in \mathbf{R}$ and $X \to Y$ is a dependency over $sort(R)$. These are sometimes referred to as *tagged* dependencies, because they are "tagged" by the relation that they apply to. The notion of satisfaction of fd's by instances over $\mathbf{R}$ is defined in the obvious way. In the remainder of this chapter, we consider only relational schemas. All can be extended easily to database schemas.

The following simple property provides the basis for the decomposition approach to schema design. Intuitively, it says that if a certain fd holds in a relation, one can store instead of the relation two projections of it, without loss of information. More precisely, the original relation can be reconstructed by joining the projections. Such joins have been termed "lossless joins" and will be discussed in some depth in Section 11.2.

PROPOSITION 8.2.2 Let I be an instance over U that satisfies $X \to Y$ and $Z = U - XY$. Then $I = \pi_{XY}(I) \bowtie \pi_{XZ}(I)$.

Proof The inclusion $I \subseteq \pi_{XY}(I) \bowtie \pi_{XZ}(I)$ holds for all instances I. For the opposite inclusion, let r be a tuple in the join. Then there are tuples $s, t \in I$ such that $\pi_{XY}(r) = \pi_{XY}(s)$ and $\pi_{XZ}(r) = \pi_{XZ}(t)$. Because $\pi_X(r) = \pi_X(t)$, and $I \models X \to Y$, $\pi_Y(r) = \pi_Y(t)$. It follows that $r = t$, so r is in I. ■

Logical Implication

In general, we may know that a set Σ of fd's is satisfied by an instance. A natural question is, What other fd's are necessarily satisfied by this instance? This is captured by the following definition.

DEFINITION 8.2.3 Let Σ and Γ be sets of fd's over an attribute set U. Then Σ (*logically*) *implies* Γ, denoted $\Sigma \models_U \Gamma$ or simply $\Sigma \models \Gamma$, if U is understood from the context, if for all relations I over U, $I \models \Sigma$ implies $I \models \Gamma$. Two sets Γ, Σ are (*logically*) *equivalent*, denoted $\Gamma \equiv \Sigma$, if $\Gamma \models \Sigma$ and $\Sigma \models \Gamma$.

EXAMPLE 8.2.4 Consider the set $\Sigma_1 = \{A \to C, B \to C, CD \to E\}$ of fd's over $\{A, B, C, D, E\}$. Then[2] a simple argument allows to show that $\Sigma_1 \models AD \to E$. In addition, $\Sigma_1 \models CDE \to C$. In fact, $\emptyset \models CDE \to C$ (where $\emptyset$ is the empty set of fd's).

Although the definition just presented focuses on fd's, this definition will be used in connection with other classes of dependencies studied here as well.

---

[2] We generally omit set braces from singleton sets of fd's.

The *fd closure* of a set Σ of fd's over an attribute set U, denoted $\Sigma^{*,U}$ or simply Σ^* if U is understood from the context, is the set

$$\{X \to Y \mid XY \subseteq U \text{ and } \Sigma \models X \to Y\}.$$

It is easily verified that for any set Σ of fd's over U and any sets $Y \subseteq X \subseteq U$, $X \to Y \in \Sigma^{*,U}$. This implies that the closure of a set of fd's depends on the underlying set of attributes. It also implies that $\Sigma^{*,U}$ has size greater than $2^{|U|}$. (It is bounded by $2^{2|U|}$ by definition.) Other properties of fd closures are considered in Exercise 8.3.

Determining Implication for fd's Is Linear Time

One of the key issues in dependency theory is the development of algorithms for testing logical implication. Although a set Σ of fd's implies an exponential (in terms of the number of attributes present in the underlying schema) number of fd's, it is possible to test whether Σ implies an fd $X \to Y$ in time that is linear in the size of Σ and $X \to Y$ (i.e., the space needed to write them).

A central concept used in this algorithm is the *fd closure* of a set of attributes. Given a set Σ of fd's over U and attribute set $X \subseteq U$, the fd closure of X under Σ, denoted $(X, \Sigma)^{*,U}$ or simply X^* if Σ and U are understood, is the set $\{A \in U \mid \Sigma \models X \to A\}$. It turns out that this set is independent of the underlying attribute set U (see Exercise 8.6).

Example 8.2.5 Recall the set Σ_1 of fd's from Example 8.2.4. Then $A^* = AC$, $(AB)^* = ABC$, and $(AD)^* = ACDE$. The family of subsets X of U such that $X^* = X$ is $\{\emptyset, C, D, E, AC, BC, CE, DE, ABC, ACE, ADE, BCE, BDE, CDE, ABCE, ACDE, BCDE, ABCDE\}$.

The following is easily verified (see Exercise 8.4):

Lemma 8.2.6 Let Σ be a set of fd's and $X \to Y$ an fd. Then $\Sigma \models X \to Y$ iff $Y \subseteq X^*$.

Thus testing whether $\Sigma \models X \to Y$ can be accomplished by computing X^*. The following algorithm can be used to compute this set.

Algorithm 8.2.7

Input: a set Σ of fd's and a set X of attributes.

Output: the closure X^* of X under Σ.

1. *unused* := Σ;
2. *closure* := X;
3. repeat until no further change:
   if $W \to Z \in$ *unused* and $W \subseteq$ *closure* then
      i. *unused* := *unused* $- \{W \to Z\}$;
      ii. *closure* := *closure* $\cup Z$
4. output *closure*.

PROPOSITION 8.2.8 On input Σ and X, Algorithm 8.2.7 computes $(X, \Sigma)^*$.

Proof Let U be a set of attributes containing the attributes occurring in Σ or X, and let *result* be the output of the algorithm. Using properties established in Exercise 8.5, an easy induction shows that $result \subseteq X^*$.

For the opposite inclusion, note first that for attribute sets Y, Z, if $Y \subseteq Z$ then $Y^* \subseteq Z^*$. Because $X \subseteq result$, it now suffices to show that $result^* \subseteq result$. It is enough to show that if $A \in U - result$, then $\Sigma \not\models result \to A$. To show this, we construct an instance I over U such that $I \models \Sigma$ but $I \not\models result \to A$ for $A \in U - result$. Let $I = \{s, t\}$, where $\pi_{result}(s) = \pi_{result}(t)$ and $s(A) \neq t(A)$ for each $A \in U - result$. (Observe that this uses the fact that the domain has at least two elements.) Note that, by construction, for each fd $W \to Z \in \Sigma$, if $W \subseteq result$ then $Z \subseteq result$. It easily follows that $I \models \Sigma$. Furthermore, for $A \in U - result$, $s(A) \neq t(A)$, so $I \not\models result \to A$. Thus $\Sigma \not\models result \to A$, and $result^* \subseteq result$. ■

The algorithm provides the means for checking whether a set of dependencies implies a single dependency. To test implication of a *set* of dependencies, it suffices to test independently the implication of each dependency in the set. In addition, one can check that the preceding algorithm runs in time $O(n^2)$, where n is the length of Σ and X. As shown in Exercise 8.7, this algorithm can be improved to linear time. The following summarizes this development.

THEOREM 8.2.9 Given a set Σ of fd's and a single fd σ, determine whether $\Sigma \models \sigma$ can be decided in linear time.

Several interesting properties of fd-closure sets are considered in Exercises 8.11 and 8.12.

Axiomatization for fd's

In addition to developing algorithms for determining logical implication, the second fundamental theme in dependency theory has been the development of inference rules, which can be used to generate symbolic proofs of logical implication. Although the inference rules do not typically yield the most efficient mechanisms for deciding logical implication, in many cases they capture concisely the essential properties of the dependencies under study. The study of inference rules is especially intriguing because (as will be seen in the next section) there are several classes of dependencies for which there is no finite set of inference rules that characterizes logical implication.

Inference rules and algorithms for testing implication provide alternative approaches to showing logical implication between dependencies. In general, the existence of a finite set of inference rules for a class of dependencies is a stronger property than the existence of an algorithm for testing implication. It will be shown in Chapter 9 that

- the existence of a finite set of inference rules for a class of dependencies implies the existence of an algorithm for testing logical implication; and

- there are dependencies for which there is no finite set of inference rules but for which there is an algorithm to test logical implication.

We now present the inference rules for fd's.

FD1: (reflexivity) If $Y \subseteq X$, then $X \rightarrow Y$.

FD2: (augmentation) If $X \rightarrow Y$, then $XZ \rightarrow YZ$.

FD3: (transitivity) If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

The variables X, Y, Z range over sets of attributes. The first rule is sometimes called an *axiom* because it is degenerate in the sense that no fd's occur in the antecedent.

The inference rules are used to form proofs about logical implication between fd's, in a manner analogous to the proofs found in mathematical logic. It will be shown that the resulting proof system is "sound" and "complete" for fd's (two classical notions to be recalled soon). Before formally presenting the notion of proof, we give an example.

Example 8.2.10 The following is a proof of $AD \rightarrow E$ from the set Σ_1 of fd's of Example 8.2.4.

σ_1 :	$A \rightarrow C$	$\in \Sigma_1$,
σ_2 :	$AD \rightarrow CD$	from σ_1 using FD2,
σ_3 :	$CD \rightarrow E$	$\in \Sigma_1$,
σ_4 :	$AD \rightarrow E$	from σ_2 and σ_3 using FD3.

Let U be a set of attributes. A *substitution* for an inference rule ρ (relative to U) is a function that maps each variable appearing in ρ to a subset of U, such that each set inclusion indicated in the antecedent of ρ is satisfied by the associated sets. Now let Σ be a set of fd's over U and σ an fd over U. A *proof* of σ from Σ using the set $\mathcal{I}$ = {FD1, FD2, FD3} is a sequence of fd's $\sigma_1, \ldots, \sigma_n = \sigma$ $(n \geq 1)$ such that for each $i \in [1, n]$, either

(a) $\sigma_i \in \Sigma$, or

(b) there is a substitution for some rule $\rho \in \mathcal{I}$ such that σ_i corresponds to the consequent of ρ, and such that for each fd in the antecedent of ρ the corresponding fd is in the set $\{\sigma_j \mid 1 \leq j < i\}$.

The fd σ is *provable* from Σ using $\mathcal{I}$ (relative to U), denoted $\Sigma \vdash_{\mathcal{I}} \sigma$ or $\Sigma \vdash \sigma$ if $\mathcal{I}$ is understood from the context, if there is a proof of σ from Σ using $\mathcal{I}$.

Let $\mathcal{I}$ be a set of inference rules. Then

$\mathcal{I}$ is *sound* for logical implication of fd's if $\Sigma \vdash_{\mathcal{I}} \sigma$ implies $\Sigma \models \sigma$,

$\mathcal{I}$ is *complete* for logical implication of fd's if $\Sigma \models \sigma$ implies $\Sigma \vdash_{\mathcal{I}} \sigma$.

We will generalize these definitions to other dependencies and other sets of inference rules.

In general, a finite sound and complete set of inference rules for a class $\mathcal{C}$ of dependencies is called a (finite) *axiomatization* of $\mathcal{C}$. In such a case, $\mathcal{C}$ is said to be (finitely) *axiomatizable*.

We now state the following:

THEOREM 8.2.11 The set {FD1, FD2, FD3} is sound and complete for logical implication of fd's.

Proof Suppose that Σ is a set of fd's over an attribute set U. The proof of soundness involves a straightforward induction on proofs $\sigma_1, \ldots, \sigma_n$ from Σ, showing that $\Sigma \models \sigma_i$ for each $i \in [1, n]$ (see Exercise 8.5).

For the proof of completeness, we show that $\Sigma \models X \to Y$ implies $\Sigma \vdash X \to Y$. As a first step, we show that $\Sigma \vdash X \to X^*$ using an induction based on Algorithm 8.2.7. In particular, let $closure_i$ be the value of *closure* after i iterations of step 3 for some fixed execution of that algorithm on input Σ and X. We set $closure_0 = X$. Suppose inductively that a proof $\sigma_1, \ldots, \sigma_{k_i}$ of $X \to closure_i$ has been constructed. [The case for $i = 0$ follows from FD1.] Suppose further that $W \to Z$ is chosen for the $(i+1)^{\text{st}}$ iteration. It follows that $W \subseteq closure_i$ and $closure_{i+1} = closure_i \cup Z$. Extend the proof by adding the following steps:

σ_{k_i+1}	=	$W \to Z$	in Σ
σ_{k_i+2}	=	$closure_i \to W$	by FD1
σ_{k_i+3}	=	$closure_i \to Z$	by FD3
σ_{k_i+4}	=	$closure_i \to closure_{i+1}$	by FD2
σ_{k_i+5}	=	$X \to closure_{i+1}$	by FD3

At the completion of this construction we have a proof $\sigma_1, \ldots, \sigma_n$ of $X \to X^*$. By Lemma 8.2.6, $Y \subseteq X^*$. Using FD1 and FD3, the proof can be extended to yield a proof of $X \to Y$. ■

Other inference rules for fd's are considered in Exercise 8.9.

Armstrong Relations

In the proof of Proposition 8.2.8, an instance I is created such that $I \models \Sigma$ but $I \not\models X \to A$. Intuitively, this instance witnesses the fact that $\Sigma \not\models X \to A$. This raises the following natural question: Given a set Σ of fd's over U, is there a *single* instance I that satisfies Σ and that violates every fd not in Σ^*? It turns out that for each set of fd's, there is such an instance; these are called *Armstrong relations*.

PROPOSITION 8.2.12 If Σ is a set of fd's over U, then there is an instance I such that, for each fd σ over U, $I \models \sigma$ iff $\sigma \in \Sigma^*$.

Crux Suppose first that $\Sigma \not\models \emptyset \to A$ for any A (i.e., $\emptyset^* = \emptyset$). For each set $X \subseteq U$ satisfying $X = X^*$, choose an instance $I_X = \{s_X, t_X\}$ such that $s_X(A) = t_X(A)$ iff $A \in X$. In addition, choose these instances so that $adom(I_X) \cap adom(I_Y) = \emptyset$ for $X \neq Y$. Then

$$\cup\{I_X \mid X \subset U \text{ and } X = X^*\}$$

is an Armstrong relation for Σ.

If $\emptyset^* \neq \emptyset$, then the instances I_X should be modified so that $\pi_A(I_X) = \pi_A(I_Y)$ for each X, Y and $A \in \emptyset^*$. ■

In some applications, the domains of certain attributes may be finite (e.g., *Sex* conventionally has two values, and *Grade* typically consists of a finite set of values). In such cases, the construction of an Armstrong relation may not be possible. This is explored in Exercise 8.13.

Armstrong relations can be used in practice to assist the user in specifying the fd's for a particular application. An interactive, iterative specification process starts with the user specifying a first set of fd's. The system then generates an Armstrong relation for the fd's, which violates all the fd's not included in the specification. This serves as a worst-case counterexample and may result in detecting additional fd's whose satisfaction should be required.

8.3 Join and Multivalued Dependencies

The second kind of simple dependency studied in this chapter is *join dependency* (jd), which is intimately related to the join operator of the relational algebra. As mentioned in Section 8.1, a basic motivation for join dependency stems from its usefulness in connection with relation decomposition. This section also discusses *multivalued dependency* (mvd), an important special case of join dependency that was historically the first to be introduced.

The central results and tools for studying jd's are different from those for fd's. It has been shown that there is no sound and complete set of inference rules for jd's analogous to those for fd's. (An axiomatization for a much larger family of dependencies will be presented in Chapter 10.) In addition, as shown in the following section, logical implication for jd's is decidable. The complexity of implication is polynomial for a fixed database schema but becomes NP-hard if the schema is considered part of the input. (An exact characterization of the complexity remains open.)

The following section also presents an interesting correspondence between mvd's and acyclic join dependencies (i.e., those based on joins that are acyclic in the sense introduced in Chapter 6).

A major focus of the current section is on mvd's; this is because of several positive results that hold for them, including axiomatizability of fd's and mvd's considered together.

Join Dependency and Decomposition

Before defining join dependency, we recall the definition of natural join. For attribute set U, sets $X_1, \ldots, X_n \subseteq U$, and instances I_j over X_j for $j \in [1, n]$, the (*natural*) *join* of the I_j's is

$$\bowtie_{j=1}^{n} \{I_j\} = \{s \text{ over } \cup X_j \mid \pi_{X_j}(s) \in I_j \text{ for each } j \in [1, n]\}.$$

A join dependency is satisfied by an instance I if it is equal to the join of some of its projections.

DEFINITION 8.3.1 A *join dependency* (jd) over attribute set U is an expression of the form $\bowtie[X_1, \ldots, X_n]$, where $X_1, \ldots, X_n \subseteq U$ and $\cup_{i=1}^{n} X_i = U$. A relation I over U *satisfies* $\bowtie[X_1, \ldots, X_n]$ if $I = \bowtie_{j=1}^{n} \{\pi_{X_j}(I)\}$.

A jd σ is *n-ary* if the number of attribute sets involved in σ is n. As discussed earlier, the relation *Showings* of Fig. 8.1 satisfies the 2-ary jd

$$\bowtie[\{Theater, Screen, Title\}, \{Theater, Snacks\}].$$

The 2-ary jd's are also called *multivalued dependencies* (mvd's). These are often denoted in a style reminiscent of fd's.

DEFINITION 8.3.2 If U is a set of attributes, then a *multivalued dependency* (mvd) over U is an expression of the form $X \twoheadrightarrow Y$, where $X, Y \subseteq U$. A relation I over U *satisfies* $X \twoheadrightarrow Y$ if $I \models \bowtie[XY, X(U - Y)]$.

In the preceding definition, it would be equivalent to write $\bowtie[XY, (U - Y)]$; we choose the foregoing form to emphasize the importance of X. For instance, the jd

$$\bowtie[\{Theater, Screen, Title\}, \{Theater, Snack\}]$$

can be written as an mvd using

$$Theater \twoheadrightarrow Screen, Title, \quad \text{or equivalently,} \quad Theater \twoheadrightarrow Snack.$$

Exercise 8.16 explores the original definition of satisfaction of an mvd.

Figure 8.2 shows a relation schema SDT and an instance that satisfies a 3-ary jd. This relation focuses on snacks, distributors, and theaters. We assume for this example that a tuple (s, d, p, t) is in SDT if the conjunction of the following predicates is true:

$P_1(s, d, p)$:	Snack s is supplied by distributor d at price p.
$P_2(d, t)$:	Theater t is a customer of distributor d.
$P_3(s, t)$:	Snack s is bought by theater t.

Under these assumptions, each instance of SDT must satisfy the jd:

$$\bowtie[\{Snack, Distributor, Price\}, \{Distributor, Theater\}, \{Snack, Theater\}].$$

For example, this holds for the instance in Fig. 8.2. Note that if tuple ⟨coffee, Smart, 2.35, Cinoche⟩ were removed, then the instance would no longer satisfy the jd because ⟨coffee, Smart, 2.35⟩, ⟨coffee, Cinoche⟩, and ⟨Smart, Cinoche⟩ would remain in the appropriate projections. We also expect the instances of SDT to satisfy $Snack, Distributor \rightarrow Price$.

It can be argued that schema *SDT* with the aforementioned constraint is unnatural in the following sense. Intuitively, if we choose such a schema, the presence of a tuple

SDT	*Snack*	*Distributor*	*Price*	*Theater*
	coffee	Smart	2.35	Rex
	coffee	Smart	2.35	Le Champo
	coffee	Smart	2.35	Cinoche
	coffee	Leclerc	2.60	Cinoche
	wine	Smart	0.80	Rex
	wine	Smart	0.80	Cinoche
	popcorn	Leclerc	5.60	Cinoche

Figure 8.2: Illustration of join dependency

$\langle s, d, p, t\rangle$ seems to indicate that t buys s from d. If we wish to record just the information about who buys what, who sells what, and who sells to whom, a more appropriate schema would consist of three relations $SD[Snack, Distributor, Price]$, $ST[Snack, Theater]$, and $DT[Distributor, Theater]$ corresponding to the three sets of attributes involved in the preceding jd. The jd then guarantees that no information is lost in the decomposition because the original relation can be reconstructed by joining the projections.

Join Dependencies and Functional Dependencies

The interaction of fd's and jd's is important in the area of schema design and user interfaces to the relational model. Although this is explored in more depth in Chapter 11, we present here one of the first results on the interaction of the two kinds of dependencies.

Proposition 8.3.3 Let U be a set of attributes, $\{X, Y, Z\}$ be a partition of U, and Σ be a set of fd's over U. Then $\Sigma \models \bowtie[XY, XZ]$ iff either $\Sigma \models X \to Y$ or $\Sigma \models X \to Z$.

Crux Sufficiency follows immediately from Proposition 8.2.2. For necessity, suppose that Σ does not imply either of the fd's. Then $Y - X^* \neq \emptyset$ and $Z - X^* \neq \emptyset$, say $C \in Y - X^*$ and $C' \in Z - X^*$. Consider the two-element instance $I = \{u, v\}$ where, $u(A) = v(A) = 0$ if A is in X^* and $u(A) = 0$, $v(A) = 1$ otherwise. Clearly, I satisfies Σ and one can verify that $\pi_{XY}(I) \bowtie \pi_{XZ}(I)$ contains a tuple w with $w(C) = 0$ and $w(C') = 1$. Thus w is not in I, so I violates $\bowtie[XY, XZ]$. ■

Axiomatizations

As will be seen later (Theorem 8.4.12), there is a decision procedure for jd's in isolation, and for jd's and fd's considered together. Here we consider axiomatizations, first for jd's in isolation and then for fd's and mvd's taken together.

We state first the following result without proof.

Theorem 8.3.4 There is no axiomatization for the family of jd's.

In contrast, there is an axiomatization for the class of fd's and multivalued dependencies. Note first that implication for fd's is independent of the underlying set of attributes (i.e., if $\Sigma \cup \{\sigma\}$ is a set of fd's over U and $V \supseteq U$, then $\Sigma \models \sigma$ relative to U iff $\Sigma \models \sigma$ relative to V; see Exercise 8.6). An important difference between fd's and mvd's is that this is not the case for mvd's. Thus the inference rules for mvd's must be used in connection with a fixed underlying set of attributes, and a variable (denoted U) referring to this set is used in one of the rules.

The following lists the four rules for mvd's alone and an additional pair of rules needed when fd's are incorporated.

MVD0: (complementation) If $X \twoheadrightarrow Y$, then $X \twoheadrightarrow (U - Y)$.

MVD1: (reflexivity) If $Y \subseteq X$, then $X \twoheadrightarrow Y$.

MVD2: (augmentation) If $X \twoheadrightarrow Y$, then $XZ \twoheadrightarrow YZ$.

MVD3: (transitivity) If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow (Z - Y)$.

FMVD1: (conversion) If $X \to Y$, then $X \twoheadrightarrow Y$.

FMVD2: (interaction) If $X \twoheadrightarrow Y$ and $XY \to Z$, then $X \to (Z - Y)$.

Theorem 8.3.5 The set {FD1, FD2, FD3, MVD0, MVD1, MVD2, MVD3, FMVD1, FMVD2} is sound and complete for logical implication of fd's and mvd's considered together.

Crux Soundness is easily verified. For completeness, let an underlying set U of attributes be fixed, and assume that $\Sigma \not\vdash \sigma$, where $\sigma = X \to Y$ or $\sigma = X \twoheadrightarrow Y$.

The *dependency set* of X is $dep(X) = \{Y \subseteq U \mid \Sigma \vdash X \twoheadrightarrow Y\}$. One first shows that

1. $dep(X)$ is a Boolean algebra of sets for U.

That is, it contains U and is closed under intersection, union, and difference (see Exercise 8.17). In addition,

2. for each $A \in X^+$, $\{A\} \in dep(X)$,

where X^+ denotes $\{A \in U \mid \Sigma \vdash X \to A\}$.

A *dependency basis* of X is a family $\{W_1, \ldots, W_m\} \subseteq dep(X)$ such that (1) $\cup_{i=1}^{n} W_i = U$; (2) $W_i \neq \emptyset$ for $i \in [1, n]$; (3) $W_i \cap W_j = \emptyset$ for $i, j \in [1, n]$ with $i \neq j$; and (4) if $W \in dep(X)$, $W \neq \emptyset$, and $W \subseteq W_i$ for some $i \in [1, n]$, then $W = W_i$. One then proves that

3. there exists a unique dependency basis of X.

Now construct an instance I over U that contains all tuples t satisfying the following conditions:

(a) $t(A) = 0$ for each $A \in X^+$.

(b) If W_i is in the dependency basis and $W_i \neq \{A\}$ for each $A \in X^+$, then $t(B) = 0$ for all $B \in W_i$ or $t(B) = 1$ for all $B \in W_i$.

It can be shown that $I \models \Sigma$ but $I \not\models \sigma$ (see Exercise 8.17). ■

This easily implies the following (see Exercise 8.18):

COROLLARY 8.3.6 The set {MVD0, MVD1, MVD2, MVD3} is sound and complete for logical implication of mvd's considered alone.

8.4 The Chase

This section presents the chase, a remarkable tool for reasoning about dependencies that highlights a strong connection between dependencies and tableau queries. The discussion here is cast in terms of fd's and jd's, but as will be seen in Chapter 10, the chase generalizes naturally to a broader class of dependencies. At the end of this section, we explore important applications of the chase technique. We show how it can also be used to determine logical implication between sets of dependencies and to optimize conjunctive queries.

The following example illustrates an intriguing connection between dependencies and tableau queries.

EXAMPLE 8.4.1 Consider the tableau query (T, t) shown in Fig. 8.3(a). Suppose the query is applied only to instances I satisfying some set Σ of fd's and jd's. The chase is based on the following simple idea. If ν is a valuation embedding T into an instance I satisfying Σ, $\nu(T)$ must satisfy Σ. Valuations that do not satisfy Σ are therefore of no use. The chase is a procedure that eliminates the useless valuations by changing (T, t) itself so that T, viewed as an instance, satisfies Σ. We will show that the tableau query resulting from the chase is then equivalent to the original on instances satisfying Σ. As we shall see, this can be used to optimize queries and test implication of dependencies.

Let us return to the example. Suppose first that $\Sigma = \{B \to D\}$. Suppose (T, t) is applied to an instance I satisfying Σ. In each valuation embedding T into I, it must be the case that z and z' are mapped to the same constant. Thus in this context one might as well replace T by the tableau where $z = z'$. This transformation is called "applying the fd $B \to D$" to (T, t). It is easy to see that the resulting tableau query is in fact equivalent to the identity, because T contains an entire row of distinguished variables.

Consider next an example involving both fd's and jd's. Let Σ consist of the following two dependencies over $ABCD$: the jd $\bowtie[AB, BCD]$ and the fd $A \to C$. In this example we argue that for each I satisfying these dependencies, $(T, t)(I) = I$ or, in other words, in the context of input instances that satisfy the dependencies, the query (T, t) is equivalent to the identity query $(\{t\}, t)$.

Let I be an instance over $ABCD$ satisfying the two dependencies. We first explain why $(T, t)(I) = (T', t)(I)$ for the tableau query (T', t) of Fig. 8.3(b). It is clear that $(T', t)(I) \subseteq (T, t)(I)$, because T' is a superset of T. For the opposite inclusion, suppose that ν is a valuation for T with $\nu(T) \subseteq I$. Then, in particular, both $\nu(\langle w, x, y, z' \rangle)$ and $\nu(\langle w', x, y', z \rangle)$ are in I. Because $I \models \bowtie[AB, BCD]$, it follows that $\nu(\langle w, x, y', z \rangle) \in I$. Thus $\nu(T') \subseteq I$ and $\nu(t) \in (T', t)(I)$. The transformation from (T, t) to (T', t) is termed "applying the jd $\bowtie[AB, BCD]$," because T' is the result of adding a member of $\pi_{AB}(T) \bowtie$

	A	B	C	D
T	w	x	y	z'
	w'	x	y'	z
t	w	x	y	z

(a) The tableau query (T, t)

	A	B	C	D
T'	w	x	y	z'
	w'	x	y'	z
	w	x	y'	z
t	w	x	y	z

(b) (One) result of applying the jd $\bowtie[AB, BCD]$

	A	B	C	D
T''	w	x	y	z'
	w'	x	y	z
	w	x	y	z
t	w	x	y	z

(c) Result of applying the fd $A \to C$

Figure 8.3: Illustration of the chase

$\pi_{BCD}(T)$ to T. We shall see that, by repeated applications of a jd, one can eventually "force" the tableau to satisfy the jd.

The tableau T'' of Fig. 8.3(c) is the result of chasing (T', t) with the fd $A \to C$ (i.e., replacing all occurrences of y' by y). We now argue that $(T', t)(I) = (T'', t)(I)$. First, by Theorem 6.2.3, $(T', t)(I) \supseteq (T'', t)(I)$ because there is a homomorphism from (T', t) to (T'', t). For the opposite inclusion, suppose now that $\nu(T') \subseteq I$. This implies that ν embeds the first tuple of T'' into I. In addition, because $\nu(\langle w, x, y, z'\rangle)$ and $\nu(\langle w, x, y', z\rangle)$ are in I and $I \models A \to C$, it follows that $\nu(y) = \nu(y')$. Thus $\nu(\langle w', x, y, z\rangle) = \nu(\langle w', x, y', z\rangle) \in I$, and $\nu(\langle w, x, y, z\rangle) = \nu(\langle w, x, y', z\rangle) \in I$, [i.e., ν embeds the second and third tuples of T'' into I, such that $\nu(T'') \subseteq I$]. Note that (T'', t) is the result of identifying a pair of variables that caused a violation of $A \to C$ in T'. We will see that by repeated applications of an fd, one can eventually "force" a tableau to satisfy the fd. Note that in this case, chasing with respect to $A \to C$ has no effect before chasing with respect to $\bowtie[AB, BCD]$.

Finally, note that by the Homomorphism Theorem 6.2.3 of Chapter 6, $(T'', t) \equiv (\{t\}, t)$. It follows, then, that for all instances I that satisfy $\{A \to C, \bowtie[AB, BCD]\}$, (T, t) and $(\{t\}, t)$ yield the same answer.

Defining the Chase

As seen in Example 8.4.1, the chase relates to equivalence of queries over a family of instances satisfying certain dependencies. For a family $\mathcal{F}$ of instances over $\mathbf{R}$, we say that q_1 is *contained* in q_2 *relative to* $\mathcal{F}$, denoted $q_1 \subseteq_{\mathcal{F}} q_2$, if $q_1(\mathbf{I}) \subseteq q_2(\mathbf{I})$ for each instance $\mathbf{I}$ in $\mathcal{F}$. We are particularly interested in families $\mathcal{F}$ that are defined by a set Σ of dependencies (in the current context, fd's and jd's). Let Σ be a set of (functional and join) dependencies over $\mathbf{R}$. The *satisfaction family* of Σ, denoted $sat(\mathbf{R}, \Sigma)$ or simply $sat(\Sigma)$ if $\mathbf{R}$ is understood from the context, is the family

$$sat(\Sigma) = \{\mathbf{I} \text{ over } \mathbf{R} \mid \mathbf{I} \models \Sigma\}.$$

Query q_1 is contained in q_2 *relative to* Σ, denoted $q_1 \subseteq_\Sigma q_2$, if $q_1 \subseteq_{sat(\Sigma)} q_2$. Equivalence relative to a family of instances ($\equiv_{\mathcal{F}}$) and to a set of dependencies ($\equiv_\Sigma$) are defined similarly.

The chase is a general technique that can be used, given a set of dependencies Σ, to transform a tableau query q into a query q' such that $q \equiv_\Sigma q'$. The chase is defined as a nondeterministic procedure based on the successive application of individual dependencies from Σ, but as will be seen this process is "Church-Rosser" in the sense that the procedure necessarily terminates with a unique end result. As a final step in this development, the chase will be used to characterize equivalence of conjunctive queries with respect to a set Σ of dependencies ($\equiv_\Sigma$).

In the following, we let R be a fixed relation schema, and we focus on sets Σ of fd's and jd's over R and tableau queries with no constants over R. The entire development can be generalized to database schemas and conjunctive queries with constants (Exercise 8.27) and to a considerably larger class of dependencies (Chapter 10).

For technical convenience, we assume that there is a total order $\leq$ on the set **var**. Let R be a fixed relation schema and suppose that (T, t) is a tableau query over R. The chase is based on the successive application of the following two rules:

fd rule: Let $\sigma = X \to A$ be an fd over R, and let $u, v \in T$ be such that $\pi_X(u) = \pi_X(v)$ and $u(A) \neq v(A)$. Let x be the lesser variable in $\{u(A), v(A)\}$ under the ordering $\leq$, and let y be the other one (i.e., $\{x, y\} = \{u(A), v(A)\}$ and $x < y$). The *result of applying* the fd σ to u, v in (T, t) is the tableau query $(\theta(T), \theta(t))$, where θ is the substitution that maps y to x and is the identity elsewhere.

jd rule: Let $\sigma = \bowtie[X_1, \ldots, X_n]$ be a jd over R, let u be a free tuple over R not in T, and suppose that $u_1, \ldots, u_n \in T$ satisfy $\pi_{X_i}(u_i) = \pi_{X_i}(u)$ for $i \in [1, n]$. Then the *result of applying* the jd σ to $(u_1, \ldots, u_n)$ in (T, t) is the tableau query $(T \cup \{u\}, t)$.

Following the lead of Example 8.4.1, the following is easily verified (see Exercise 8.24a).

Proposition 8.4.2 Suppose that Σ is a set of fd's and jd's over R, $\sigma \in \Sigma$, and q is a tableau query over R. If q' is the result of applying σ to some tuples in q, then $q' \equiv_\Sigma q$.

A *chasing sequence* of (T, t) by Σ is a (possibly infinite) sequence

$$(T, t) = (T_0, t_0), \ldots, (T_i, t_i), \ldots$$

such that for each $i \geq 0$, (T_{i+1}, t_{i+1}) (if defined) is the result of applying some dependency in Σ to (T_i, t_i). The sequence is *terminal* if it is finite and no dependency in Σ can be applied to it. The last element of the terminal sequence is called its *result*. The notion of *satisfaction* of a dependency is extended naturally to tableaux. The following is an important property of terminal chasing sequences (Exercise 8.24b).

Lemma 8.4.3 Let (T', t') be the result of a terminal chasing sequence of (T, t) by Σ. Then T', considered as an instance, satisfies Σ.

Because the chasing rules do not introduce new variables, it turns out that the chase procedure always terminates. The following is easily verified (Exercise 8.24c):

LEMMA 8.4.4 Let (T, t) be a tableau query over R and Σ a set of fd's and jd's over R. Then each chasing sequence of (T, t) by Σ is finite and is the initial subsequence of a terminal chasing sequence.

An important question now is whether the results of different terminal chasing sequences are the same. This turns out to be the case. This property of chasing sequences is called the *Church-Rosser property*. We provide the proof of the Church-Rosser property for the chase at the end of this section (Theorem 8.4.18).

Because the Church-Rosser property holds, we can define without ambiguity the result of chasing a tableau query by a set of fd's and jd's.

DEFINITION 8.4.5 If (T, t) is a tableau query over R and Σ a set of fd's and jd's over R, then the *chase* of (T, t) by Σ, denoted $chase(T, t, \Sigma)$, is the result of some (any) terminal chasing sequence of (T, t) by Σ.

From the previous discussion, $chase(T, t, \Sigma)$ can be computed as follows. The dependencies are picked in some arbitrary order and arbitrarily applied to the tableau. Applying an fd to a tableau query q can be performed within time polynomial in the size of q. However, determining whether a jd can be applied to q is NP-complete in the size of q. Thus the best-known algorithm for computing the chase is exponential (see Exercise 8.25). However, the complexity is polynomial if the schema is considered fixed.

Until now, besides the informal discussion in Section 8.1, the *chase* remains a purely syntactic technique. We next state a result that shows that the chase is in fact determined by the semantics of the dependencies in Σ and not just their syntax.

In the following proposition, recall that by definition, $\Sigma \equiv \Sigma'$ if $\Sigma \models \Sigma'$ and $\Sigma' \models \Sigma$. The proof, which we omit, uses the Church-Rosser property of the chase (see also Exercise 8.26).

PROPOSITION 8.4.6 Let Σ and Σ' be sets of fd's and jd's over R, and let (T, t) be a tableau query over R. If $\Sigma \equiv \Sigma'$, then $chase(T, t, \Sigma)$ and $chase(T, t, \Sigma')$ coincide.

We next consider several important uses of the chase that illustrate the power of this technique.

Query Equivalence

We consider first the problem of checking the equivalence of tableau queries in the presence of a set of fd's and jd's. This allows, for example, checking whether a tableau query can be replaced by a simpler tableau query when the dependencies are satisfied. Suppose now that (T', t') and (T'', t'') are two tableau queries and Σ a set of fd's and jd's such that $(T', t') \equiv_\Sigma (T'', t'')$. From the preceding development (Proposition 8.4.2), it follows that

$$chase(T', t', \Sigma) \equiv_\Sigma (T', t') \equiv_\Sigma (T'', t'') \equiv_\Sigma chase(T'', t'', \Sigma).$$

We now show that, in fact, $chase(T', t', \Sigma) \equiv chase(T'', t'', \Sigma)$. Furthermore, this condition is sufficient as well as necessary.

To demonstrate this result, we first establish the following more general fact.

THEOREM 8.4.7 Let $\mathcal{F}$ be a family of instances over relation schema R that is closed under isomorphism, and let (T_1, t_1), (T_2, t_2), (T_1', t_1'), and (T_2', t_2') be tableau queries over R. Suppose further that for $i = 1, 2$,

(a) $(T_i', t_i') \equiv_\mathcal{F} (T_i, t_i)$ and

(b) T_i', considered as an instance, is in $\mathcal{F}$.[3]

Then $(T_1, t_1) \subseteq_\mathcal{F} (T_2, t_2)$ iff $(T_1', t_1') \subseteq (T_2', t_2')$.

Proof The if direction is immediate. For the only-if direction, suppose that $(T_1, t_1) \subseteq_\mathcal{F} (T_2, t_2)$. It suffices by the Homomorphism Theorem 6.2.3 to exhibit a homomorphism that embeds (T_2', t_2') into (T_1', t_1'). Because T_1', considered as an instance, is in $\mathcal{F}$,

$$t_1' \in (T_1', t_1')(T_1') \Rightarrow t_1' \in (T_1, t_1)(T_1') \Rightarrow t_1' \in (T_2, t_2)(T_1') \Rightarrow t_1' \in (T_2', t_2')(T_1').$$

It follows that there is a homomorphism h such that $h(T_2') \subseteq T_1'$ and $h(t_2') = t_1'$. Thus $(T_1', t_1') \subseteq (T_2', t_2')$. This completes the proof. ■

Together with Lemma 8.4.3, this implies the following:

THEOREM 8.4.8 Let (T_1, t_1) and (T_2, t_2) be tableau queries over R and Σ a set of fd's and jd's over R. Then

1. $(T_1, t_1) \subseteq_\Sigma (T_2, t_2)$ iff $chase(T_1, t_1, \Sigma) \subseteq chase(T_2, t_2, \Sigma)$.
2. $(T_1, t_1) \equiv_\Sigma (T_2, t_2)$ iff $chase(T_1, t_1, \Sigma) \equiv chase(T_2, t_2, \Sigma)$.

Query Optimization

As suggested in Example 8.4.1, the chase can be used to optimize tableau queries in the presence of dependencies such as fd's and jd's. Given a tableau query (T, t) and a set Σ of fd's and jd's, $chase(T, t, \Sigma)$ is equivalent to (T, t) on all instances satisfying Σ. A priori, it is not clear that the new tableau is an improvement over the first. It turns out that the chase using fd's can never yield a more complicated tableau and, as shown in Example 8.4.1, can yield a much simpler one. On the other hand, the chase using jd's may yield a more complicated tableau, although it may also produce a simpler one.

We start by looking at the effect on tableau minimization of the chase using fd's. In the following, we denote by $min(T, t)$ the tableau resulting from the minimization of

[3] More precisely, T' considered as an instance is in $\mathcal{F}$ means that some instance isomorphic to T' is in $\mathcal{F}$.

the tableau (T, t) using the Homomorphism Theorem 6.2.3 for tableau queries, and by $|min(T, t)|$ we mean the cardinality of the tableau of $min(T, t)$.

Lemma 8.4.9 Let (T, t) be a tableau query and Σ a set of fd's. Then $|min(chase(T, t, \Sigma))| \leq |min(T, t)|$.

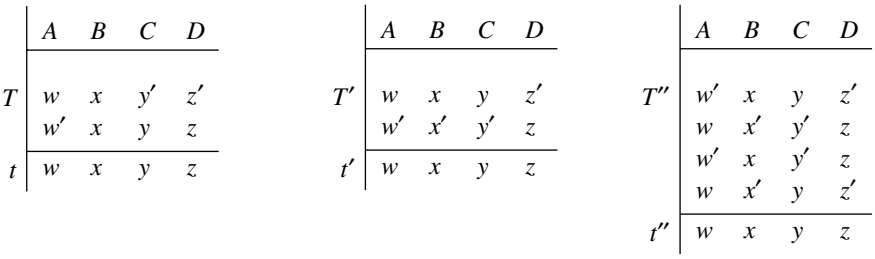
Crux By the Church-Rosser property of the chase, the order of the dependencies used in a chase sequence is irrelevant. Clearly it is sufficient to show that for each tableau query (T', t') and $\sigma \in \Sigma$, $|min(chase(T', t', \sigma))| \leq |min(T', t')|$. We can assume without loss of generality that σ is of the form $X \to A$, where A is a single attribute.

Let $(T'', t'') = chase(T', t', \{X \to A\})$, and let θ be the *chase homomorphism* of a chasing sequence for $chase(T', t', \{X \to A\})$, i.e., the homomorphism obtained by composing the substitutions used in that chasing sequence (see the proof of Theorem 8.4.18). We will use here the Church-Rosser property of the chase (Theorem 8.4.18) as well as a related property stating that the homomorphism θ, like the result, is also the same for all chase sequences (this follows from the proof of Theorem 8.4.18).

By Theorem 6.2.6, there is some $S \subseteq T'$ such that (S, t') is a minimal tableau query equivalent to (T', t'); we shall use this as the representative of $min(T', t')$. Let h be a homomorphism such that $h(T', t') = (S, t')$. Consider the mapping f on (T'', t'') defined by $f(\theta(x)) = \theta(h(x))$, where x is a variable in (T', t'). If we show that f is well defined, we are done. [If f is well defined, then f is a homomorphism from (T'', t'') to $\theta(S, t') = (\theta(S), t'')$, and so $(T'', t'') \supseteq \theta(S, t')$. On the other hand, the $\theta(S) \subseteq \theta(T') = T''$, and so $(T'', t'') \subseteq \theta(S, t')$. Thus, $(T'', t'') \equiv \theta(S, t') = \theta(min(T', t'))$, and so $|min(T'', t'')| = |min(\theta(min(T', t')))| \leq |\theta(min(T', t'))| \leq |min(T', t')|$.]

To see that f is well defined, suppose $\theta(x) = \theta(y)$. We have to show that $\theta(h(x)) = \theta(h(y))$. Consider a terminal chasing sequence of (T', t') using $X \to A$, and $(u_1, v_1), \ldots, (u_n, v_n)$ as the sequence of pairs of tuples used in the sequence, yielding the chase homomorphism θ. Consider the sequence $(h(u_1), h(v_1)), \ldots, (h(u_n), h(v_n))$. Clearly if $X \to A$ can be applied to (u, v), then it can be applied to $(h(u), h(v))$, unless $h(u(A)) = h(v(A))$. Let $(h(u_{i_1}), h(v_{i_1})), \ldots, (h(u_{i_k}), h(v_{i_k}))$ be the subsequence of these pairs for which $X \to A$ can be applied. It can be easily verified that there is a chasing sequence of $(h(T'), t')$ using $X \to A$ that uses the pairs $(h(u_{i_1}), h(v_{i_1})), \ldots, (h(u_{i_k}), h(v_{i_k}))$, with chase homomorphism θ'. Note that for all x', y', if $\theta(x') = \theta(y')$ then $\theta'(h(x')) = \theta'(h(y'))$. In particular, $\theta'(h(x)) = \theta'(h(y))$. Because $h(T') \subseteq T'$, θ' is the chase homomorphism of a chasing sequence $\sigma_1, \ldots, \sigma_k$ of (T', t'). Let θ'' be the chase homomorphism formed from a terminal chasing sequence that extends $\sigma_1, \ldots, \sigma_k$. Then $\theta''(h(x)) = \theta''(h(y))$. Finally, by the uniqueness of the chase homomorphism, $\theta'' = \theta$, and so $\theta(h(x)) = \theta(h(y))$ as desired. This concludes the proof. ■

It turns out that jd's behave differently than fd's with respect to minimization of tableaux. The following shows that the chase using jd's may yield simpler but also more complicated tableaux.



	A	B	C	D
T	w	x	y′	z′
	w′	x	y	z
t	w	x	y	z

(a) The tableau query (T, t)

	A	B	C	D
T′	w	x	y	z′
	w′	x′	y′	z
t′	w	x	y	z

(b) The tableau query (T', t')

	A	B	C	D
T″	w′	x	y	z′
	w	x′	y′	z
	w′	x	y′	z
	w	x′	y	z′
t″	w	x	y	z

(c) The tableau query $chase(T', t', \{\bowtie[AB, CD]\})$

Figure 8.4: Minimization and the chase using jd's

EXAMPLE 8.4.10 Consider the tableau query (T, t) shown in Fig. 8.4(a) and the jd $\sigma = \bowtie[AB, BCD]$. Clearly (T, t) is minimal, so $|min(T, t)| = 2$. Next consider $chase(T, t, \sigma)$. It is easy to check that $\langle w, x, y, z\rangle \in chase(T, t, \sigma)$, so $chase(T, t, \sigma)$ is equivalent to the identity and

$$|min(chase(T, t, \sigma))| = 1.$$

Next let (T', t') be the tableau query in Fig. 8.4(b) and $\sigma = \bowtie[AB, CD]$. Again (T', t') is minimal. Now $chase(T', t', \sigma)$ is represented in Fig. 8.4(c) and is minimal. Thus

$$|min(chase(T', t', \sigma))| = 4 > |min(T', t')|.$$

Despite the limitations illustrated by the preceding example, the chase in conjunction with tableau minimization provides a powerful optimization technique that yields good results in many cases. This is illustrated by the following example and by Exercise 8.28.

EXAMPLE 8.4.11 Consider the SPJ expression

$$q = \pi_{AB}(\pi_{BCD}(R) \bowtie \pi_{ACD}(R)) \bowtie \pi_{AD}(R),$$

where R is a relation with attributes $ABCD$. Suppose we wish to optimize the query on databases satisfying the dependencies

$$\Sigma = \{B \to D, D \to C, \bowtie[AB, ACD]\}.$$

The tableau (T, t) corresponding to q is represented in Fig. 8.5(a). Note that (T, t) is minimal. Next we chase (T, t) using the dependencies in Σ. The chase using the fd's in Σ does not change (T, t), which already satisfies them. The chase using the jd

	A	B	C	D
T	w'	x	y'	z'
	w	x'	y'	z'
	w	x''	y''	z
t	w	x	y	z

(a) The tableau query (T, t) corresponding to q

	A	B	C	D
T''	w'	x	y'	z'
	w	x'	y'	z
	w	x''	y'	z
t''	w	x	y	z

(c) The tableau query $(T'', t'') = chase(T', t', \{B \to D, D \to C\})$

	A	B	C	D
T'	w'	x	y'	z'
	w	x'	y'	z'
	w	x''	y''	z
	w	x'	y''	z
	w	x''	y'	z'
t'	w	x	y	z

(b) The tableau query $(T', t') = chase(T, t, \{\bowtie[AB, ACD]\})$

	A	B	C	D
T'''	w'	x	y'	z
	w	x'	y'	z
t'''	w	x	y	z

(d) The tableau query $(T''', t''') = min(T'', t'')$

Figure 8.5: Optimization of SPJ expressions by tableau minimization and the chase

$\bowtie[AB, ACD]$ yields the tableau (T', t') in Fig. 8.5(b). Now the fd's can be applied to (T', t') yielding the tableau (T'', t'') in Fig. 8.5(c). Finally (T'', t'') is minimized to (T''', t''') in Fig. 8.5(d). Note that (T''', t''') satisfies Σ, so the chase can no longer be applied. The SPJ expression corresponding to (T''', t''') is $\pi_{ABD}(\pi_{BCD}(R) \bowtie \pi_{ACD}(R))$. Thus, the optimization of q resulted in saving one join operation. Note that the new query is not simply a subexpression of the original. In general, the shape of queries can be changed radically by the foregoing procedure.

The Chase and Logical Implication

We consider a natural correspondence between dependency satisfaction and conjunctive query containment. This correspondence uses tableaux to represent dependencies. We will see that the chase provides an alternative point of view to dependency implication.

First consider a jd $\sigma = \bowtie[X_1, \ldots, X_n]$. It is immediate to see that an instance I satisfies σ iff $q_\sigma(I) \subseteq q_{id}(I)$, where

$$q_\sigma = [X_1] \bowtie \cdots \bowtie [X_n]$$

and q_{id} is the identity query. Both q_σ and q_{id} are PSJR expressions. We can look at alternative formalisms for expressing q_σ and q_{id}. For instance, the *tableau query of* σ is (T_σ, t), where for some $t_1, \ldots, t_n$,

- t is a free tuple over R with a distinct variable for each coordinate,
- $T_\sigma = \{t_1, \ldots, t_n\}$,
- $\pi_{X_i}(t_i) = \pi_{X_i}(t)$ for $i \in [1, n]$, and
- the other coordinates of the t_i's hold distinct variables.

It is again easy to see that $q_\sigma = (T_\sigma, t)$, so $I \models \sigma$ iff $(T_\sigma, t)(I) \subseteq (\{t\}, t)(I)$.

For fd's, the situation is only slightly more complicated. Consider an fd $\sigma' = X \to A$ over U. It is easy to see that $I \models \sigma'$ iff $(T_{\sigma'}, t_{\sigma'})(I) \subseteq (T_{\sigma'}, t'_{\sigma'})(I)$, where

	X	A	$(U - AX)$
$T_{\sigma'}$	u	x	v_1
	u	x'	v_2
$t_{\sigma'}$	x	x'	

	X	A	$(U - AX)$
	u	x	v_1
	u	x'	v_2
$t'_{\sigma'}$	x	x	

where u, v_1, v_2 are vectors of distinct variables and x, x' are distinct variables occurring in none of these vectors. The *tableau query* of σ' is $(T_{\sigma'}, t_{\sigma'})$.

Again observe that $(T_{\sigma'}, t_{\sigma'})$, (T_σ, t_σ) can be expressed as PSJR expressions, so fd satisfaction also reduces to containment of PSJR expressions. It will thus be natural to look more generally at all dependencies expressed as containment of PSJR expressions. In Chapter 10, we will consider the general class of *algebraic dependencies* based on containment of these expressions.

Returning to the chase, we next use the tableau representation of dependencies to obtain a characterization of logical implication (Exercise 8.29). This result is generalized by Corollary 10.2.3.

Theorem 8.4.12 Let Σ and $\{\sigma\}$ be sets of fd's and jd's over relation schema R, let (T_σ, t_σ) be the tableau query of σ, and let T be the tableau in $chase(T_\sigma, t_\sigma, \Sigma)$. Then $\Sigma \models \sigma$ iff

(a) $\sigma = X \to A$ and $|\pi_A(T)| = 1$, that is, the projection over A of T is a singleton; or

(b) $\sigma = \bowtie[X_1, \ldots, X_n]$ and $t_\sigma \in T$.

This implies that determining logical implication for jd's alone, and for fd's and jd's taken together, is decidable. On the other hand, tableau techniques are also used to obtain the following complexity results for logical implication of jd's (see Exercise 8.30).

THEOREM 8.4.13

(a) Testing whether a jd and an fd imply a jd is NP-complete.

(b) Testing whether a set of mvd's implies a jd is NP-hard.

Acyclic Join Dependencies

In Section 6.4, a special family of joins called acyclic was introduced and was shown to enjoy a number of desirable properties. We show now a connection between those results, join dependencies, and multivalued dependencies.

A jd $\bowtie[X_1, \ldots, X_n]$ is *acyclic* if the hypergraph corresponding to $[X_1, \ldots, X_n]$ is acyclic (as defined in Section 6.4).

Using the chase, we show here that a jd is acyclic iff it is equivalent to a set of mvd's. The discussion relies on the notation and techniques developed in the discussion of acyclic joins in Section 6.4.

We shall use the following lemma.

LEMMA 8.4.14 Let $\sigma = \bowtie\mathbf{X}$ be a jd over U, and let $X, Y \subseteq U$ be disjoint sets. Then the following are equivalent:

(i) $\sigma \models X \twoheadrightarrow Y$;

(ii) there is no $X_i \in \mathbf{X}$ such that $X_i \cap Y \neq \emptyset$ and $X_i \cap (U - XY) \neq \emptyset$;

(iii) Y is a union of connected components of the hypergraph $\mathbf{X}|_{U-X}$.

Proof Let $Z = U - XY$. Let τ denote the mvd $X \twoheadrightarrow Y$, and let (T_τ, t_τ) be the tableau query corresponding to τ. Let $T_\tau = \{t_Y, t_Z\}$ where $t_Y[XY] = t_\tau[XY]$ and $t_Z[XZ] = t_\tau[XZ]$ and distinct variables are used elsewhere in t_Y and t_Z.

We show now that (i) implies (ii). By Theorem 8.4.12, $t_\tau \in T = chase(T_\tau, t_\tau, \sigma)$. Let $X_i \in \mathbf{X}$. Suppose that t is a new tuple created by an application of σ during the computation of T. Then $t[X_i]$ agrees with $t'[X_i]$ for some already existing tuple. An induction implies that $t_\tau[X_i] = t_Y[X_i]$ or $t_\tau[X_i] = t_Z[X_i]$. Because t_Y and t_Z agree only on X, this implies that X_i cannot intersect with both Y and Z.

That (ii) implies (iii) is immediate. To see that (iii) implies (i), consider an application of the jd $\bowtie\mathbf{X}$ on T_τ, where $X_i \in \mathbf{X}$ is associated with t_Y if $X_i - X \subseteq Y$, and X_i is associated with t_Z otherwise. This builds the tuple t_τ, and so by Theorem 8.4.12, $\sigma \models X \twoheadrightarrow Y$. ■

We now have the following:

THEOREM 8.4.15 A jd σ is acyclic iff there is a set Σ of mvd's that is equivalent to σ.

Proof (**only if**) Suppose that $\sigma = \bowtie\mathbf{X}$ over U is acyclic. By Theorem 6.4.5, this implies that the output of the GYO algorithm on $\mathbf{X}$ is empty. Let $X_1, \ldots, X_n$ be an enumeration of $\mathbf{X}$ in the order of an execution of the GYO algorithm. In particular, X_i is an ear of the hypergraph formed by $\{X_{i+1}, \ldots, X_n\}$.

For each $i \in [1, n-1]$, let $P_i = \cup_{j\in[1,i]} X_j$ and $Q_i = \cup_{j\in[i+1,n]} X_j$. Let $\Sigma = \{[P_i \cap Q_i] \twoheadrightarrow Q_i \mid i \in [1, n-1]\}$. By Lemma 8.4.14 and the choice of sequence $X_1, \ldots, X_n$, $\sigma \models \Sigma$. To show that $\Sigma \models \sigma$, we construct a chasing sequence of (T_σ, t_σ) using Σ that yields t_σ. This chase shall inductively produce a sequence $t_1, \ldots, t_n$ of tuples, such that $t_i[P_i] = t_\sigma[P_i]$ for $i \in [1, n]$.

We begin by setting t_1 to be the tuple of T_σ that corresponds to X_1. Then $t_1[P_1] = t_\sigma[P_1]$ because $P_1 = X_1$. More generally, given t_i with $i \geq 1$, the mvd $[P_i \cap Q_i] \twoheadrightarrow Q_i$ on t_i and the tuple corresponding to X_{i+1} can be used to construct tuple t_{i+1} with the desired property. The final tuple t_n constructed by this process is t_σ, and so $\Sigma \models \sigma$ as desired.

(**if**) Suppose that $\sigma = \bowtie\mathbf{X}$ over U is equivalent to the set Σ of mvd's but that σ is not acyclic. From the definition of acyclic, this implies that there is some $W \subseteq U$ such that $\mathbf{Y} = \mathbf{X}|_W$ has no articulation pair. Without loss of generality we assume that $\mathbf{Y}$ is connected.

Let $\mathbf{Y} = \{Y_1, \ldots, Y_m\}$. Suppose that $s_1, \ldots$ are the tuples produced by some chasing sequence of (T_σ, t_σ). We argue by induction that for each $k \geq 1$, $s_k[W] \in \pi_W(T_\sigma)$. Suppose otherwise, and let s_k be the first where this does not hold. Suppose that s_k is the result of applying an mvd $X \twoheadrightarrow Y$ in Σ. Without loss of generality we assume that $X \cap Y = \emptyset$. Let $Z = U - XY$. Because s_k results from $X \twoheadrightarrow Y$, there are two tuples s' and s'' either in T_σ or already produced, such that $s_k[XY] = s'[XY]$ and $s_k[XZ] = s''[XZ]$. Because s_k is chosen to be least, there are tuples t_i and t_j in T_σ, which correspond to X_i and X_j, respectively, such that $s'[W] = t_i[W]$ and $s''[W] = t_j[W]$.

Because t_i and t_j correspond to X_i and X_j, for each attribute $A \in U$ we have $t_i[A] = t_j[A]$ iff $A \in X_i \cap X_j$. Thus $X \cap W \subseteq X_i \cap X_j$.

Because $s_k[W] \neq t_i[W]$, $W - XZ \neq \emptyset$, and because $s_k[W] \neq t_j[W]$, $W - XY \neq \emptyset$. Now, by Lemma 8.4.14, because $X \twoheadrightarrow Y$ is implied by σ, there is no $X_k \in \mathbf{X}$ such that $X_k \cap Y \neq \emptyset$ and $X_k \cap Z \neq \emptyset$. It follows that $\mathbf{Y}|_{W-X}$ is disconnected. Finally, let $Y = X_i \cap W$ and $Y' = X_j \cap W$. Because $X \cap W \subseteq X_i \cap X_j$, it follows that $Y \cap Y'$ is an articulation set for $\mathbf{Y}$, a contradiction. ■

We conclude with a complexity result about acyclic jd's. The first part follows from the proof of the preceding theorem and the fact that the GYO algorithm runs in polynomial time. The second part, stated without proof, is an interesting converse of the first part.

PROPOSITION 8.4.16

(a) There is a PTIME algorithm that, given an acyclic jd σ, produces a set of mvd's equivalent to σ.

(b) There is a PTIME algorithm that, given a set Σ of mvd's, finds a jd equivalent to Σ or determines that there is none.

The Chase Is Church-Rosser

To conclude this section, we provide the proof that the results of all terminal chasing sequences of a tableau query q by a set Σ of fd's and jd's are identical. To this end, we first introduce tools to describe correspondences between the free tuples occurring in the different elements of chasing sequences.

Let $(T, t) = (T_0, t_0), \ldots, (T_n, t_n)$ be a chasing sequence of (T, t) by Σ. Then for each $i \in [1, n]$, the *chase homomorphism* for step i, denoted θ_i, is an assignment with domain $var(T_i)$ defined as follows:

(a) If (T_{i+1}, t_{i+1}) is the result of applying the fd rule to (T_i, t_i), which replaces all occurrences of variable y by variable x, then θ_{i+1} is defined so that $\theta_{i+1}(y) = x$ and θ_{i+1} is the identity on $var(T_i) - \{y\}$.

(b) If (T_{i+1}, t_{i+1}) is the result of applying the jd rule to (T_i, t_i), then θ_{i+1} is the identity on $var(T_i)$.

The *chase homomorphism* of this chasing sequence is $\theta = \theta_1 \circ \cdots \circ \theta_n$. If $w \in (T \cup \{t\})$, then the tuple *corresponding* to w in (T_i, t_i) is $w_i = \theta_1 \circ \cdots \circ \theta_i(w)$. It may arise that $u_i = v_i$ for distinct tuples u, v in T. Observe that $\theta_1 \circ \cdots \circ \theta_i(T) \subseteq T_i$ and that, because of the jd rule, the inclusion may be strict.

We now have the following:

LEMMA 8.4.17 Suppose that $I \models \Sigma$, ν is a substitution over $var(T)$, $\nu(T) \subseteq I$, and $(T_0, t_0), \ldots, (T_n, t_n)$ is a chasing sequence of (T, t) by Σ. Then

$$\nu(w_i) = \nu(w) \text{ for each } i \in [1, n] \text{ and each } w \in (T \cup \{t\}),$$

and $\nu(T_i) \subseteq I$ for each $i \in [1, n]$.

Crux Use an induction on the chasing sequence (Exercise 8.24d). ■

Observe that this also holds if I is a tableau over R that satisfies Σ. This is used in the following result.

THEOREM 8.4.18 Let (T, t) be a tableau query over R and Σ a set of fd's and jd's over R. Then the results of all terminal chasing sequences of (T, t) by Σ are identical.

Proof Let (T', t') and (T'', t'') be the results of two terminal chasing sequences on (T, t) using Σ, and let θ', θ'' be the chase homomorphisms of these chasing sequences. For each tuple $w \in T$, let w' denote the tuple of T' that corresponds to w, and similarly for w'', T''.

By construction, $\theta''(T) \subseteq T''$ and $\theta''(t) = t''$. Because $T'' \models \Sigma$ and $\theta''(T) \subseteq T''$, $\theta''(T') \subseteq T''$ by Lemma 8.4.17 considering the chasing sequence leading to T'. The same argument shows that $\theta''(w') = w''$ for each w in T and $\theta''(t') = t''$. By symmetry, $\theta'(T'') \subseteq T'$, $\theta'(w'') = w'$ for each w in T and $\theta'(t'') = t'$.

We next prove that

(*) θ'' is an isomorphism from (T', t') to (T'', t'').

Let w'' be in T'' for some w in T. Then

$$\theta' \circ \theta''(w'') = \theta''(\theta'(w'')) = \theta''(w') = w''.$$

Observe that each variable x in $var(T'')$ occurs in w'', for some w in T. Thus $\theta' \circ \theta''$ is the identity over $var(T'')$. We therefore have

$$\theta' \circ \theta''(T'') = T''.$$

By symmetry, $\theta'' \circ \theta'$ is the identity over $var(T')$ and

$$\theta'' \circ \theta'(T') = T'.$$

Thus $|T''| = |T'|$. Because $\theta''(T') \subseteq T''$, $\theta''(T') = T''$ and θ'' is an isomorphism from (T', t') to (T'', t''), so (*) holds.

To conclude, we prove that

(**) θ'' is the identity over $var(T')$.

We first show that for each pair x, y of variables occurring in T,

(†) $\theta''(x) = \theta''(y)$ iff $\theta'(x) = \theta'(y)$.

Suppose that $\theta''(x) = \theta''(y)$. Then for some tuples $u, v \in T$ and attributes A, B, we have $u(A) = x$, $v(B) = y$ and $u''(A) = \theta''(x) = \theta''(y) = v''(B)$. Next $\theta'(x) = u'(A)$ and $\theta'(y) = v'(B)$. Because θ' is an isomorphism from (T'', t'') to (T', t') and $\theta'(u'') = u', \theta'(v'') = v'$, it follows that $u'(A) = v'(B)$. Hence $\theta'(x) = u'(A) = v'(B) = \theta'(y)$ as desired. The if direction follows by symmetry.

Now let $x \in var(T')$. To prove (**) and the theorem, it now suffices to show that $\theta''(x) = x$. Let

$$\mathcal{A}' = \{y \in var(T) \mid \theta'(y) = \theta'(x)\},$$
$$\mathcal{A}'' = \{y \in var(T) \mid \theta''(y) = \theta''(x)\}.$$

First (†) implies that $\mathcal{A}' = \mathcal{A}''$. Furthermore, an induction on the chasing sequence for (T', t') shows that for each $z \in \mathcal{A}'$, $\theta'(z)$ is the least (under the ordering on **var**) element of $\mathcal{A}'$, and similarly for (T'', t''). Thus θ' and θ'' map all elements of $\mathcal{A}'$ and $\mathcal{A}''$ to the same variable z. Because $x \in var(T')$, it follows that $z = x$ so, in particular, $\theta'(x) = \theta''(x) = x$. ■

Bibliographic Notes

On a general note, we first mention that comprehensive presentations of dependency theory can be found in [Var87, FV86]. A more dense presentation is provided in [Kan91]. Dependency theory is also the topic of the book [Tha91].

Research on general integrity constraints considered from the perspective of first-order logic is presented in [GM78]. Other early work in this framework includes [Nic78], which observes that fd's and mvd's have a natural representation in logic, and [Nic82], which

considers incremental maintanence of integrity constraints under updates to the underlying state.

Functional dependencies were introduced by Codd [Cod72b]. The axiomatization is due to [Arm74]. The problem of implication is studied in [BB79, Mai80]. Several alternative formulations of fd implication, including formulation in terms of the propositional calculus perspective (see Exercise 8.22), are mentioned in [Kan91]; they are due to [SDPF81, CK85, CKS86].

Armstrong relations were introduced and studied in [Fag82b, Fag82a, BDFS84]. Interesting practical applications of Armstrong relations are proposed in [SM81, MR85]. The idea is that, given a set Σ of fd's, the system presents an Armstrong relation for Σ with natural column entries to a user, who can then determine whether Σ includes all of the desired restrictions.

The structure of families of instances specified by a set of fd's is studied in [GZ82, Hul84].

Multivalued dependencies were discovered independently in [Zan76, Fag77b, Del78]. They were generalized in [Ris77, Nic78, ABU79]. The axiomatization of fd's and mvd's is from [BFH77]. A probabilistic view of mvd's in terms of conditional independence is presented in [PV88, Pea88]. This provides an alternative motivation for the study of such dependencies.

The issue of whether there is an axiomatization for jd's has a lengthy history. As will be detailed in Chapter 10, the family of full typed dependencies subsumes the family of jd's, and an axiomatization for these was presented in [BV84a, YP82]; see also [SU82]. More focused axiomatizations, which start with jd's and end with jd's but use slightly more general dependencies at intermediate stages, are presented in [Sci82] and [BV85]; see also [BV81b]. Reference [BV85] also develops an axiomatization for join dependencies based on Gentzen-style proofs (see, e.g., [Kle67]); proofs in this framework maintain a form of scratch paper in addition to a sequence of inferred sentences. Finally, [Pet89] settled the issue by establishing that there is no axiomatization (in the sense defined in Section 8.2) for the family of jd's.

As noted in Chapter 6, acyclic joins received wide interest in the late 1970s and early 1980s so Theorem 8.4.15 was demonstrated in [FMU82]. Proposition 8.4.16 is from [GT83].

An ancestor to the chase can be found in [ABU79]. The notion of chase was articulated in [MMS79]. Related results can be found in [MSY81, Var83]. The relationship between the chase and both tableau queries and logical implication was originally presented in [MMS79] and builds on ideas originally introduced in [ASU79b, ASU79a]. The chase technique is extended to more general dependencies in [BV84c]; see also Chapter 10. The connection between the chase and the more general theorem-proving technique of resolution with paramodulation (see [CL73]) is observed and analyzed in [BV80b]. The chase technique is applied to datalog programs in [RSUV89, RSUV93].

Exercises

Exercise 8.1 Describe the set of fd's, mvd's, and jd's that are tautologies (i.e., dependencies that are satisfied by all instances) for a relation schema R.

Exercise 8.2 Let Σ_1 be as in Example 8.2.4. Prove that $\Sigma_1 \models AD \to E$ and $\Sigma_1 \models CDE \to C$.

Exercise 8.3 Let U be a set of attributes, and let Σ, Γ be sets of dependencies over U. Show that

(a) $\Sigma \subseteq \Sigma^*$.

(b) $(\Sigma^*)^* = \Sigma^*$.

(c) If $\Gamma \subseteq \Sigma$, then $\Gamma^* \subseteq \Sigma^*$.

State and prove analogous results for fd closures of attribute sets.

Exercise 8.4 Prove Lemma 8.2.6.

Exercise 8.5 Let U be a set of attributes and Σ a set of fd's over U. Prove the soundness of FD1, FD2, FD3 and show that

$$\text{If } \Sigma \vdash X \to Y \text{ and } \Sigma \vdash X \to Z, \text{ then } \Sigma \vdash X \to YZ.$$

Exercise 8.6 Let Σ be a set of fd's over U.

(a) Suppose that $X \subseteq U$ and $U \subseteq V$. Show that $(X, \Sigma)^{*,U} = (X, \Sigma)^{*,V}$. *Hint:* Use the proof of Proposition 8.2.8.

(b) Suppose that $XY \subseteq U$, and $U \subseteq V$. Show that $\Sigma \models_U X \to Y$ iff $\Sigma \models_V X \to Y$.

♠ **Exercise 8.7** [BB79] Describe how to improve the efficiency of Algorithm 8.2.7 to linear time. *Hint:* For each unused fd $W \to Z$ in Σ, record the number attributes of W not yet in *closure*. To do this efficiently, maintain a list for each attribute A of those unused fd's of Σ for which A occurs in the left-hand side.

Exercise 8.8 Give a proof of $AB \to F$ from $\Sigma = \{AB \to C, A \to D, CD \to EF\}$ using {FD1, FD2, FD3}.

Exercise 8.9 Prove or disprove the soundness of the following rules:

FD4: (pseudo-transitivity) If $X \to Y$ and $YW \to Z$, then $XW \to Z$.

FD5: (union) If $X \to Y$ and $X \to Z$, then $X \to YZ$.

FD6: (decomposition) If $X \to YZ$, then $X \to Y$.

MVD4: (pseudo-transitivity) If $X \twoheadrightarrow Y$ and $YW \twoheadrightarrow Z$, then $XW \twoheadrightarrow Z - Y$.

MVD5: (union) If $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z$, then $X \twoheadrightarrow YZ$.

MVD6: (decomposition) If $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z$, then $X \twoheadrightarrow Y \cap Z$, $X \twoheadrightarrow Y - Z$, and $X \twoheadrightarrow Z - Y$.

bad-FD1: If $XW \to Y$ and $XY \to Z$, then $X \to (Z - W)$.

bad-MVD1: If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow Z$.

bad-FMVD1: If $X \twoheadrightarrow Y$ and $XY \to Z$, then $X \to Z$.

(The use of the hint is optional.)

Exercise 8.10 Continuing with Exercise 8.9,

(a) [BFH77] Find a two-element subset of {FD1, FD2, FD3, FD4, FD5, FD6} that is sound and complete for inferring logical implication of fd's.

(b) Prove that there is exactly one two-element subset of {FD1, FD2, FD3, FD4, FD5, FD6} that is sound and complete for inferring logical implication of fd's.

Exercise 8.11 [Arm74] Let U be a fixed set of attributes. An attribute set $X \subseteq U$ is *saturated* with respect to a set Σ of fd's over U if $X = X^*$. The family of saturated sets of Σ with respect to U is $satset(\Sigma) = \{X \subseteq U \mid X \text{ is saturated with respect to } \Sigma\}$.

(a) Show that $satset = satset(\Sigma)$ satisfies the following properties:

S1: $U \in satset$.
S2: If $Y \in satset$ and $Z \in satset$, then $Y \cap Z \in satset$.

★(b) Suppose that *satset* is a family of subsets of U satisfying properties (**S1**) and (**S2**). Prove that $satset = satset(\Gamma)$ for some set Γ of fd's over U. *Hint:* Use $\Gamma = \{Y \to Z |$ for each $X \in satset$, if $Y \subseteq X$ then $Z \subseteq X\}$.

Exercise 8.12 Let Σ and Γ be sets of fd's over U. Using the notation of Exercise 8.11,

(a) Show that $satset(\Sigma \cup \Gamma) = satset(\Sigma) \cap satset(\Gamma)$.

(b) Show that $satset(\Sigma^* \cap \Gamma^*) = satset(\Sigma) \wedge satset(\Gamma)$, where for families $\mathcal{F}, \mathcal{G}$, the *wedge* of $\mathcal{F}$ and $\mathcal{G}$ is $\mathcal{F} \wedge \mathcal{G} = \{X \cap Y \mid X \in \mathcal{F} \text{ and } Y \in \mathcal{G}\}$.

(c) For $V \subseteq U$, define $\pi_V \Sigma = \{X \to Y \in \Sigma \mid XY \subseteq V\}$. For $V \subseteq U$ characterize *satset* $(\pi_V(\Sigma^*))$ (where this family is defined with respect to V).

Exercise 8.13

(a) Exhibit a set Σ_1 of fd's over $\{A, B\}$ such that each Armstrong relation for Σ has at least three distinct values occurring in the A column. Exhibit a set Σ_2 of fd's over $\{A, B, C\}$ such that each Armstrong relation for Σ has at least four distinct values occurring in the A column.

(b) [GH83, BDFS84] Let Σ be a set of fd's over U. Recall the notion of saturated set from Exercise 8.11. For an instance I over U, the *agreement set* of I is $agset(I) = \{X \subseteq U \mid \exists\, s, t \in I$ such that $s(A) = t(A)$ iff $A \in X\}$. For a family $\mathcal{F}$ of subsets of U, the *intersection closure* of $\mathcal{F}$ is $intclo(\mathcal{F}) = \{\cap_{i=1}^{n} X_i \mid n \geq 0$ and each $X_i \in \mathcal{F}\}$ (where the empty intersection is defined to be U). Prove that I is an Armstrong relation for Σ iff $intclo(agset(I)) = satset(\Sigma)$.

Exercise 8.14 [Mai80] Let Σ be a set of fd's over U, $X \to Y \in \Sigma$, and let A be an attribute. A is *extraneous* in $X \to Y$ with respect to Σ if either

(a) $(\Sigma - \{X \to Y\}) \cup \{X \to (Y - A)\} \models X \to Y$; or

(b) $(\Sigma - \{X \to Y\}) \cup \{(X - A) \to Y\} \models X \to Y$.

Develop an $O(n^2)$ algorithm that takes as input a set Σ of fd's and produces as output a set $\Sigma' \equiv \Sigma$, where Σ' has no extraneous attributes.

Exercise 8.15 Show that there is no set Σ of jd's and fd $X \to A$ such that $\Sigma \models X \to A$. *Hint:* Show that for any instance I there exists an instance I' such that $I \subseteq I'$ and $I' \models \Sigma$. Then choose I violating $X \to A$.

Exercise 8.16 [Fag77b, Zan76] This exercise refers to the original definition of mvd's. Let U be a set of attributes and $X, Y \subseteq U$. Given an instance I over U and a tuple $x \in \pi_X(I)$, the *image*

of x on Y in I is the set $image_Y(x, I) = \pi_Y(\sigma_{X=x}(I))$ of tuples over Y. Prove that $I \models X \twoheadrightarrow Y$ iff

$$\text{for each } x \in \pi_X(I) \text{ and each } z \in image_Z(x, I),\ image_Y(x, I) = image_Y(xz, I),$$

where $Z = U - XY$ and xz denotes the tuple w over XZ such that $\pi_X(w) = x$ and $\pi_Z(w) = z$.

★ **Exercise 8.17** [BFH77] Complete the proof of Theorem 8.3.5. *Hint:* Of course, the inference rules can be used when reasoning about I. The following claims are also useful:

Claim 1: If $A \in X^+$, then $I \models \emptyset \to A$.

Claim 2: If $A, B \in W_i$ for some $i \in [1, n]$, then $I \models A \to B$.

Claim 3: For each $i \in [1, n]$, $I \models \emptyset \twoheadrightarrow W_i$.

Exercise 8.18 Prove Corollary 8.3.6.

Exercise 8.19 [Kan91] Consider the following set of inference rules:

MVD7: $X \twoheadrightarrow U - X$.

MVD8: If $Y \cap Z = \emptyset$, $X \twoheadrightarrow Y$, and $Z \twoheadrightarrow W$, then $X \twoheadrightarrow W - Y$.

FMVD3: If $Y \cap Z = \emptyset$, $X \twoheadrightarrow Y$, and $Z \to W$, then $X \to Y \cap W$.

Prove that {MVD7, MVD2, MVD8} are sound and complete for inferring implication for mvd's, and that {FD1, FD2, FD3, MVD7, MVD2, MVD8, FMVD1, FMVD3} are sound and complete for inferring implication for fd's and mvd's considered together.

Exercise 8.20 [Bee80] Let Σ be a set of fd's and mvd's, and let $m(\Sigma) = \{X \twoheadrightarrow Y \mid X \twoheadrightarrow Y \in \Sigma\} \cup \{X \twoheadrightarrow A \mid A \in Y \text{ for some } X \to Y \in \Sigma\}$. Prove that

(a) $\Sigma \models X \to Y$ implies $m(\Sigma) \models X \twoheadrightarrow Y$; and

(b) $\Sigma \models X \twoheadrightarrow Y$ iff $m(\Sigma) \models X \twoheadrightarrow Y$.

Hint: For (b) do an induction on proofs using the inference rules.

Exercise 8.21 For sets Σ and Γ of dependencies over U, Σ *implies* Γ *for two-element instances*, denoted $\Sigma \models_2 \Gamma$, if for each instance I over U with $|I| \leq 2$, $I \models \Sigma$ implies $I \models \Gamma$.

(a) [SDPF81] Prove that if $\Sigma \cup \{\sigma\}$ is a set of fd's and mvd's, then $\Sigma \models_2 \sigma$ iff $\Sigma \models \sigma$.

(b) Prove that the equivalence of part (a) does not hold if jd's are included.

(c) Exhibit a jd σ such that there is no set Σ of mvd's with $\sigma \equiv \Sigma$.

♠ **Exercise 8.22** [SDPF81] This exercise develops a close connection between fd's and mvd's, on the one hand, and a fragment of propositional logic, on the other. Let U be a fixed set of attributes. We view each attribute $A \in U$ as a propositional variable. For the purposes of this exercise, a *truth assignment* is a mapping $\xi : U \to \{T, F\}$ (where T denotes *true* and F denotes *false*). Truth assignments are extended to mappings on subsets X of U by $\xi(X) = \wedge_{A \in X} \xi(A)$. A truth assignment ξ *satisfies* an fd $X \to Y$, denoted $\xi \models X \to Y$, if $\xi(X) = T$ implies $\xi(Y) = T$. It satisfies an mvd $X \twoheadrightarrow Y$, denoted $\xi \models X \twoheadrightarrow Y$, if $\xi(X) = T$ implies that either $\xi(Y) = T$ or $\xi(U - Y) = T$. Given a set $\Sigma \cup \{\sigma\}$ of fd's and mvd's, Σ implies σ *in the propositional calculus*, denoted $\Sigma \models_{\text{prop}} \sigma$, if for each truth assignment ξ, $\xi \models \Sigma$ implies $\xi \models \sigma$. Prove that for all sets $\Sigma \cup \{\sigma\}$ of fd's and mvd's, $\Sigma \models \sigma$ iff $\Sigma \models_{\text{prop}} \sigma$.

★ **Exercise 8.23** [Bis80] Exhibit a set of inference rules for mvd's that are sound and complete in the context in which an underlying set of attributes is not fixed.

♠ **Exercise 8.24**

(a) Prove Proposition 8.4.2.

(b) Prove Lemma 8.4.3.

(c) Prove Lemma 8.4.4. What is the maximum size attainable by the tableau in the result of a terminal chasing sequence?

(d) Prove Lemma 8.4.17.

♠ **Exercise 8.25**

(a) Describe a polynomial time algorithm for computing the chase of a tableau query by Σ, assuming that Σ contains only fd's.

(b) Show that the problem of deciding whether a jd can be applied to a tableau query is NP-complete if the schema is considered variable, and polynomial if the schema is considered fixed. *Hint:* Use Exercise 6.16.

(c) Prove that it is NP-hard, given a tableau query (T, t) and a set Σ of fd's and jd's, to compute $chase(T, t, \Sigma)$ (this assumes that the schema is part of the input and thus not fixed).

(d) Describe an exponential time algorithm for computing the chase by a set of fd's and jd's. (Again the schema is not considered fixed.)

Exercise 8.26 Prove Proposition 8.4.6. *Hint:* Rather than modifying the proof of Theorem 8.4.18, prove as a lemma that if $\Sigma \models \sigma$, then $chase(T, t, \Sigma) = chase(T, t, \Sigma \cup \{\sigma\})$.

Exercise 8.27

(a) Verify that the results concerning the chase generalize immediately to the context in which database schemas as opposed to relation schemas are used.

(b) Describe how to generalize the chase to tableau in which constants occur, and state and prove the results about the chase and tableau queries. *Hint:* If the chase procedure attempts to equate two distinct constants (a situation not occurring before), we obtain a particular new tableau, called T_{false}, which corresponds to the query producing an empty result on all input instances.

Exercise 8.28 For each of the following relation schemas R, SPJ expressions q over R, and dependencies Σ over R, simplify q knowing that it is applied only to instances over R satisfying Σ. Use tableau minimization and the chase.

(a) $sort(R) = ABC$, $q = \pi_{AC}(\pi_{AB}(\sigma_{A=2}(R) \bowtie \pi_{BC}(R)) \bowtie \pi_{AB}(\sigma_{B=8}(R) \bowtie \pi_{BC}(R))$, $\Sigma = \{A \to C, B \to C\}$

(b) $sort(R) = ABCD$, $q = \pi_{BC}(R) \bowtie \pi_{ABD}(R)$, $\Sigma = \{B \twoheadrightarrow CD, B \twoheadrightarrow D\}$

(c) $sort(R) = ABCD$, $q = \pi_{ABD}(R) \bowtie \pi_{AC}(R)$, $\Sigma = \{A \to B, B \twoheadrightarrow C\}$.

♠ **Exercise 8.29** Prove Theorem 8.4.12.

♠ **Exercise 8.30** Prove Theorem 8.4.13(a) [BV80a] and Theorem 8.4.13(b) [FT83].

Exercise 8.31 [MMS79] Describe an algorithm based on the chase for

(a) computing the closure of an attribute set X under a set Σ of fd's and jd's (where the notion of closure is extended to include all fd's implied by Σ); and

(b) computing the dependency basis (see Section 8.3) of a set X of attributes under a set Σ of fd's and jd's (where the notion of dependency basis is extended to include fd's in the natural manner).

Exercise 8.32 [GH86] Suppose that the underlying domain **dom** has a total order $\leq$. Let $U = \{A_1, \ldots, A_n\}$ be a set of attributes. For each $X \subseteq U$, define the partial order $\leq_X$ over the set of tuples of X by $t \leq_X t'$ iff $t(A) \leq t'(A)$ for each $A \in X$. A *sort set dependency* (SSD) over U is an expression of the form $s(X)$, where $X \subseteq U$. An instance I over U satisfies $s(X)$, denoted $I \models s(X)$, if $\leq_X$ is a total order on $\pi_X(I)$.

(a) Show that the following set of inference rules is sound and complete for finite logical implication between SSDs:
SSD1: If A is an attribute, then $s(A)$.
SSD2: If $s(X)$ and $Y \subseteq X$, then $s(Y)$.
SSD3: If $s(X)$, $s(Y)$ and $s(X \triangle Y)$, then $s(XY)$ [where $X \triangle Y$ denotes $(X - Y) \cup (Y - X)$, i.e., the symmetric difference of X and Y].

(b) Exhibit a polynomial time algorithm for inferring logical implication between sets of SSDs.

(c) Describe how SSDs might be used in connection with indexes.