Integrity = Validity + Completeness

AMIHAI MOTRO
University of Southern California

Database integrity has two complementary components: *validity*, which guarantees that all false information is excluded from the database, and *completeness*, which guarantees that all true information is included in the database. This article describes a uniform model of integrity for relational databases, that considers both validity and completeness. To a large degree, this model subsumes the prevailing model of integrity (i.e., integrity constraints). One of the features of the new model is the determination of the integrity of answers issued by the database system in response to user queries. To users, answers that are accompanied with such detailed certifications of their integrity are more meaningful. First, the model is defined and discussed. Then, a specific mechanism is described that implements this model. With this mechanism, the determination of the integrity of an answer is a process analogous to the determination of the answer itself.

Categories and Subject Descriptors: H.2.0 **[Database Management]**: General—*security, integrity, and protection*; H.2.1 **[Database Management]**: Logical Design—*data models*; H.2.4 **[Database Management]**: Systems—*query processing*

General Terms: Design, Languages, Theory

Additional Key Words and Phrases: Closed world assumption, completeness, integrity, integrity constraints, metarelation, relational algebra, relational database, relational view, validity

1. INTRODUCTION

When one receives information from a database (or, for that matter, from any other source), the question of the *integrity* of the information that is delivered immediately comes to one's mind. Usually, this question has two parts: (1) Is the information valid? and (2) Is it complete? For example, when a prospective traveler requests a database to list all flights from Los Angeles to New York, he is concerned with the *validity* of the information he receives (does each listing indeed represent an actual flight from Los Angeles to New York?), but also with its *completeness* (are there any other flights from Los Angeles to New York that were not listed?).

In other words, answers have integrity if they contain *the whole truth* (completeness) and *nothing but the truth* (validity).

This work was supported in part by NSF Grant No. IRI-8609912 and by an Amoco Foundation Engineering Faculty Grant. Submitted for publication in ACM Transactions on Database Systems. Second Revision, January 1989.

Author's address: Computer Science Department, University of Southern California, University Park, Los Angeles, CA 90089-0782

Until now, all efforts to enhance the integrity of relational databases have been within the framework of *integrity constraints*. In general, integrity constraints are formulas in predicate calculus that express relationships that must be satisfied by the database [1, 2]. Assuming that initially a database satisfies the constraints, update requests thereafter are accepted only if they do not violate any of the constraints.

Such constraints *enhance* the integrity of a database (and hence the integrity of its answers), but they cannot *ensure* it. Thus, it is not possible to express integrity constraints that ensure that the information about each flight from Los Angeles to New York is valid, or that all actual flights from Los Angeles to New York are included.

Another potential problem with this integrity model is that constraints are applied *selectively* to prevent certain undesirable data relationships, but when answers are issued, there is no way to distinguish the information monitored by constraints from the rest. For example, a database with flight information may not have a constraint that prevents flights of a particular airline that depart on different days from having the same flight number. The answer to a query about flights from Los Angeles to New York via TWA may then show flight number TW-391 as departing on two different days. And yet, this information is issued with the same guarantee as information that is indeed monitored by integrity constraints.

For reasons of efficiency, most database systems do not permit constraints that are arbitrary predicate calculus formulas. Usually, only very specific types of formulas are permitted. For example, UNIFY [10] permits only *referential* constraints, and INGRES [8] permits only *range* constraints. Thus, this integrity model remains largely unimplemented.

In this article we describe a model of integrity control that addresses these problems. New kinds of integrity constraints, called *validity constraints* and *completeness constraints* are introduced.[1] Roughly, these constraints are intended to ensure the validity and the completeness of the information in the database, and hence the validity and completeness of each answer. Together, they ensure the integrity of the information in the database. In general, the enforcement of these constraints requires human supervision. However, a particular family of constraints will be defined that can be enforced by the database system itself. As we shall show, to a large degree, the new integrity model *subsumes* the integrity model that employs "traditional" integrity constraints.

In addition, the new integrity model is designed to *certify* the integrity of answers. For each answer issued in response to a query, it determines (1) whether the answer is *valid* (the information is ensured to be correct), or only *partially valid* (specified portions are ensured to be correct); and (2) whether the answer is *complete* (the information is ensured to include all the real world occurrences), or only *partially complete* (specified portions are ensured to include all the real world occurrences). In effect, the database is *qualifying* its answers, by the scope of its knowledge (as defined by the integrity constraints).

[1] Interestingly, *integrity* expresses both validity and completeness, and we shall continue to use this term.

In addition to the general model, we describe a specific mechanism that implements it efficiently. For a more restricted, yet powerful, family of integrity constraints we develop a representation that stores constraints in *meta-relations* that mirror the actual database relations. And we extend standard relational algebra operations to these meta-relations. When a query is presented to the database, it is performed *both* on the relations, resulting in a set of tuples, and on the meta-relations, resulting in a set of constraints. The derived constraints assert the integrity of the derived answer. Thus, the determination of the integrity of an answer is a process analogous to the determination of the answer itself.

As an example, consider a database with information about flights, and assume it satisfies two integrity constraints that guarantee valid information about nonstop flights and complete information about flights via domestic carriers. The answer to a query to retrieve all flights from Los Angeles to New York that depart on Sundays is certified to be valid with regard to nonstop flights and complete with regard to domestic carriers. Clearly, answers that are accompanied by such "certification of quality" are more meaningful.

Central to our approach is the view that validity and completeness are two complementary components of integrity, and we define a uniform framework that accommodates both components and allows us to elucidate their duality. Completeness constraints were first introduced in [5], and an application of completeness constraints in verifying user presuppositions is described in [6].

The concept of information completeness is related to the *Closed World Assumption* (*CWA*) [7]. Under this assumption, a database contains all the occurrences of data that it attempts to model. More accurately, the CWA states that if a fact is not included in the database (and cannot be inferred from it), then it is false. The CWA is usually made on the database *as a whole*. In practice, however, this assumption is not realistic, as most databases include at least some information that is possibly incomplete. In other words, in reality the CWA can only be made on some *subsets* of the database. Levesque [3] argues the same point, and offers a solution based on extending predicate calculus with formulas that express what the database knows (*K formulas*). Roughly speaking, our completeness constraints assert that certain database subsets are "closed world." More accurately, completeness constraints correspond to predicates whose interpretations must contain *all* the tuples that represent real world relationships; validity constraints correspond to predicates whose interpretations must contain *only* tuples that represent real world relationships.

This article is divided into two parts. In the first part (Sections 2, 3, and 4) we define a model for integrity of relational databases. Section 2 defines the basic concepts of integrity constraints and database integrity. In particular, it distinguishes between "absolute" integrity (the database is a perfect model of the real world), and "qualified" integrity (the database is a perfect model of certain portions of the real world). The integrity of databases can be violated either through unwarranted changes to the database, or as a result of changes in the real world. Monitoring database integrity is discussed in Section 3. An important goal of integrity control is to determine the integrity of each answer. The integrity of answers is defined and discussed in Section 4. In the second part (Sections 5, 6, and 7) we concentrate on a particular mechanism for implementing this model

with a more restricted family of constraints. Section 5 describes a representation for these constraints in meta-relations, Section 6 defines extensions to relational algebra operations for operating on meta-relations, and Section 7 illustrates the mechanism with examples. Section 8 concludes this article with a brief summary and discussion of several further issues.

2. DATABASE VALIDITY AND COMPLETENESS

To formalize the concepts of validity and completeness in relational databases, we shall assume the existence of a hypothetical database that captures a designated environment of the real world perfectly. The stored database is then an *approximation* of this hypothetical database.

We assume the following definition of a relational database [4]. A *relation scheme* R is a finite set of *attributes* $A_1, \ldots, A_m$. With each attribute A_i a set of values D_i, called the *domain* of A_i, is associated (domains are nonempty, finite, or countably infinite sets). A *relation* on the relation scheme R is a subset of the Cartesian product of the domains associated with the attributes of R. A *database scheme* $\mathscr{R}$ is a set of relation schemes $R_1, \ldots, R_n$. A database instance $\mathbf{D}$ of the database scheme $\mathscr{R}$ is a set of relations $R_1(\mathbf{D}), \ldots, R_n(\mathbf{D})$, where each $R_i(\mathbf{D})$ is a relation on the relation scheme R_i.

Assume a database scheme $\mathscr{R}$. We consider three different database instances of $\mathscr{R}$. $\mathbf{U}$ is the instance of all *possible* tuples ("the universe"); for every relation scheme $R \in \mathscr{R}$, the relation $R(\mathbf{U})$ is defined as the Cartesian product of the domains associated with the attributes of R. At any particular moment some tuples in $\mathbf{U}$ are *true* (represent associations which currently hold in the real world), and the others are *false*. $\mathbf{W}$ is the instance of all *true* tuples ("the real world"); for every relation scheme $R \in \mathscr{R}$, the relation $R(\mathbf{W})$ is defined as the tuples of $R(\mathbf{U})$ that are true. Finally, $\mathbf{D}$ is the instance of all *stored* tuples ("the database"); for every relation scheme $R \in \mathscr{R}$, the relation $R(\mathbf{D})$ is defined as the tuples of $R(\mathbf{U})$ that are stored in the database. The relationships between $\mathbf{U}$, $\mathbf{W}$, and $\mathbf{D}$ are illustrated in Figure 1. Note that both $\mathbf{W}$ and $\mathbf{D}$ may change with time. Thus, their definitions are always relative to a particular moment in time.

We define the *difference* between two database instances as the database obtained by taking the difference between the corresponding relations, and the *empty* database instance as the instance whose relations are all empty. The discrepancy between $\mathbf{D}$ and $\mathbf{W}$ is of two kinds: $\mathbf{D} - \mathbf{W}$ is composed entirely of database tuples that are false; $\mathbf{W} - \mathbf{D}$ is composed entirely of true tuples that are not in the database. If $\mathbf{D} - \mathbf{W} = \varnothing$, then $\mathbf{D}$ is *valid* (with respect to $\mathbf{W}$). If $\mathbf{W} - \mathbf{D} = \varnothing$, then $\mathbf{D}$ is *complete* (with respect to $\mathbf{W}$). If $\mathbf{D}$ is both valid and complete (with respect to $\mathbf{W}$), then $\mathbf{D} = \mathbf{W}$ and $\mathbf{D}$ has *integrity* (with respect to $\mathbf{W}$). A database instance that has integrity is a perfect model of the real world.

The above definitions compare the *entire* instances $\mathbf{D}$ and $\mathbf{W}$, so they can be seen as definitions of total integrity. The following definitions are more refined, in that they compare only *selective views* of $\mathbf{D}$ and $\mathbf{W}$.

A *view* V is an expression in the relation schemes of $\mathscr{R}$ that defines a new relation scheme, and for each database instance $\mathbf{D}$ defines a unique relation on this scheme denoted $V(\mathbf{D})$.

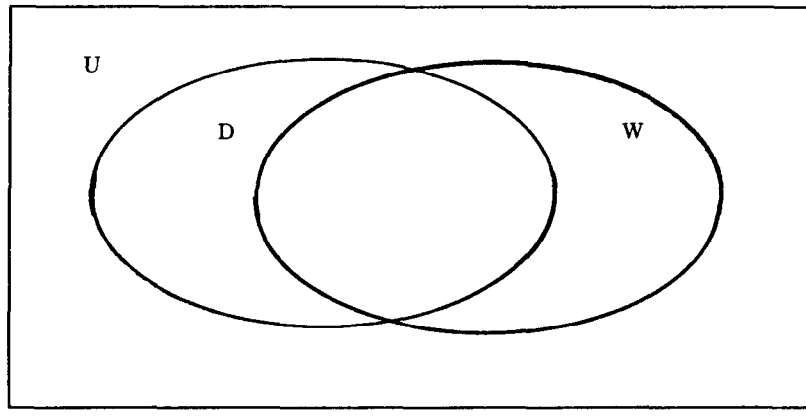

Fig. 1. The universe of all possible tuples **U**, the real world **W**, and a database **D**.

We define a view V to be *valid* in **D** (with respect to **W**), if $V(\mathbf{D}) \subseteq V(\mathbf{W})$. That is, the view of the database is contained in the view of the real world. We define a view V to be *complete* in **D** (with respect to **W**), if $V(\mathbf{W}) \subseteq V(\mathbf{D})$. That is, the view of the database contains the view of the real world. A view that is both valid and complete in **D** (with respect to **W**) is said to have *integrity* in **D** (with respect to **W**).

To illustrate these concepts, Figure 2 shows two views called V and C. V is valid in D if the top shaded area is empty. C is *complete* in **D** if the bottom shaded area is empty.

We now modify the definition of a database scheme to include (in addition to $\mathscr{R}$) two sets of views: $\mathscr{V}$ is a set of views of $\mathscr{R}$ called *validity constraints*, and $\mathscr{C}$ is a set of views of $\mathscr{R}$ called *completeness constraints*. Finally, we modify the definition of a database instance **D**, requiring also that every view in $\mathscr{V}$ be valid in **D** (with respect to **W**), and every view in $\mathscr{C}$ be complete in **D** (with respect to **W**).

Note that the new definition of a database instance subsumes the earlier definition of integrity: If we select $\mathscr{V}$ and $\mathscr{C}$ to be $\mathscr{R}$, then instances of this database will have integrity.

Validity constraints are concerned with false information that *must not* be part of a valid database. For example, assume that it is desirable to ensure the validity of the set of nonstop flights from Los Angeles to New York. The appropriate validity constraint would be "nonstop flights from Los Angeles to New York." A database satisfies this validity constraint if it does not include any false nonstop flights from Los Angeles to New York.

Completeness constraints are concerned with true information that *must* be part of a complete database. For example, assume that it is desirable to ensure the completeness of the set of flights from Los Angeles to New York via domestic carriers. The appropriate completeness constraint would be "flights from Los Angeles to New York via domestic carriers." A database satisfies this completeness constraint if it includes every true flight from Los Angeles to New York via a domestic carrier.

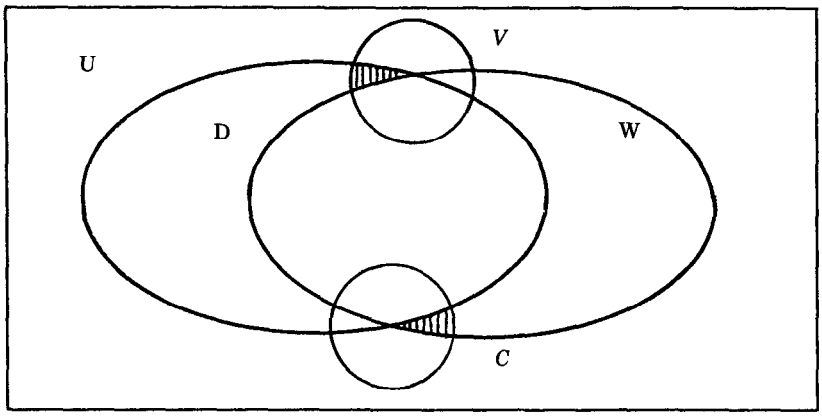

Fig. 2. A valid view V and a complete view C.

3. MONITORING DATABASE INTEGRITY

If both the database and the real world environment it models are *static*, then a database with integrity will always retain its integrity. In general, however, real world environments are subject to changes, and databases must change accordingly. Such changes may *violate* the integrity of databases. Violations of integrity are of four different types. The following analysis is demonstrated with the validity constraint "nonstop flights from Los Angeles to New York" and the completeness constraint "flights from Los Angles to New York via domestic carriers."

A validity constraint is violated whenever a tuple is in its view of the database but not in its view of the real world. A validity constraint currently satisfied by the database may be violated in one of two ways:

V1. The database is updated by adding a tuple to the view of the database that is false (is not in the view of the real world). For example, a nonexisting nonstop flight from Los Angeles to New York is added to the database.

V2. The real world changes and a tuple is deleted from the view of the real world, which is stored (is in the view of the database). For example, a nonstop flight from Los Angeles to New York which is stored in the database is canceled by the carrier.

A completeness constraint is violated whenever a tuple is in its view of the world but not in its view of the database. A completeness constraint currently satisfied by the database may be violated in one of two ways:

C1. The database is updated by deleting a tuple from the view of the database that is true (is in the view of the world). For example, an existing flight from Los Angeles to New York via a domestic carrier is deleted from the database.

C2. The real world changes and a tuple is added to the view of the real world, which is not stored (is not in the view of the database). For example, a new flight from Los Angeles to New York is started by a domestic carrier.

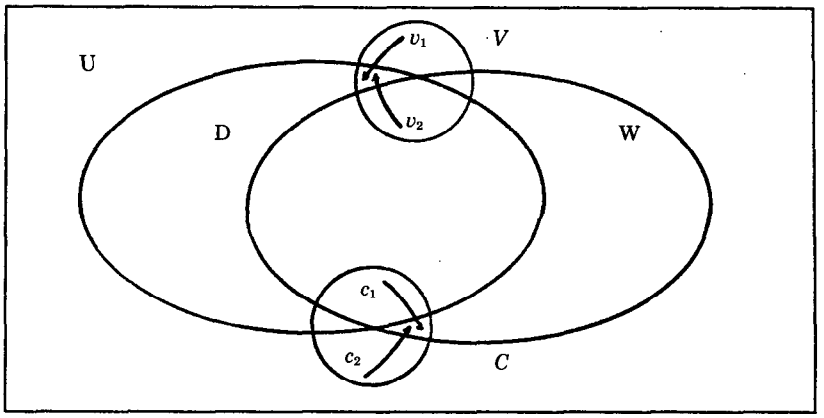

Fig. 3. Violations of constraints.

Figure 3 illustrates the migrations of tuples involved in the four types of integrity violations. These four types are of two categories: violations caused by unwarranted modifications to the database (V1 and C1), and violations caused by continuing changes in the real world (V2 and C2).

All four types of integrity violation involve testing whether a tuple is true (is in the view of **W**) or false (is not in the view of **W**). However, the real world **W** is a hypothetical database whose actual membership is usually not known. Consequently, it may not be feasible to determine whether a tuple is true or not.

Therefore, in the general case, it would be the responsibility of the *database administrator* (i.e., an authorized person) to monitor the integrity of the database. To detect violations of the former category (types V1 and C1), the database administrator must authorize any updates to the database that add to the views of validity constraints (verify that they are true), or delete from the views of completeness constraints (verify that they are false). To detect violations of the latter category (types V2 and C2), the database administrator must track the real world and update the database to delete from the views of validity constraints tuples that become false, and add to the views of completeness constraints tuples that become true.

Thus, human supervision is necessary for two reasons: (1) A database system cannot determine whether a tuple is true or false; if it could, then violations of types V1 and C1 could be detected automatically (the system would then reject the update attempts); (2) A database system cannot sense changes in the real world; if it could, then violations of types V2 and C2 could be detected automatically (the system would then update the database accordingly).

Consider now the implementation of validity constraints with the following restriction: *Views of validity constraints never include true tuples* (i.e., for each $V \in \mathscr{V}$, $V(\mathbf{W}) = \varnothing$). This restriction has two consequences. First, the situation described in V1 is easy to detect, as it is only necessary to determine whether the update will add a tuple to the view of the validity constraint (i.e., it is not necessary to verify that the tuple is also false!). Therefore, all updates that add tuples to views of validity constraints are rejected by the system. As an example,

assume that flights from Los Angeles to Tel Aviv always require a stop-over. The validity constraint "nonstop flights from Los Angeles to Tel Aviv" would then specify only false tuples. Thus, any update that would result in a nonstop flight from Los Angeles to Tel Aviv may be rejected right away. Second, if there are never any true tuples in the view of a validity constraint, then the situation described in V2 cannot occur. Thus, the restriction on the type of validity constraints relieves the database administrator from the need to monitor the real world for tuples in the view of validity constraints that become false and should be deleted from the database.

In conclusion, validity constraints that never specify true tuples may be monitored by the system itself: Every attempt to update the database is rejected if it would add to the view of such a constraint.

Consider now an implementation of completeness constraints with the dual restriction: *Views of completeness constraints never include false tuples* (i.e., for each $C \in \mathscr{C}$, $C(\mathbf{W}) = C(\mathbf{U})$). The consequences of this restriction are analogous. First, the situation described in C1 is easy to detect, as it is only necessary to determine whether the update will delete a tuple from the view of the completeness constraint (i.e., it is not necessary to verify that the tuple is also true!). Therefore, all updates that delete tuples from views of completeness constraints are rejected by the system. As an example, assume that TWA has daily flights from Los Angeles to New York. The completeness constraint "days on which TWA flys from Los Angeles to New York" would then specify only true tuples. Thus, any update that would result in unavailability of a TWA flight from Los Angeles to New York on a particular day may be rejected right away. Second, if there are never any false tuples in the view of a completeness constraint, then the situation described in C2 cannot occur. Thus, the restriction on the type of completeness constraints relieves the database administrator from the need to monitor the real world for tuples in the view of completeness constraints that become true and should be added to the database.

In conclusion, completeness constraints that never specify false tuples may be monitored by the system itself: Every attempt to update the database is rejected if it would delete from the view of such a constraint.

Of course, the database system must be told which validity and completeness constraints are of these restricted kinds, so that it would monitor updates to the views that correspond to these constraints. Also, it should be noted that, in some cases, the assumptions that validity constraints never specify true tuples and completeness constraints never specify false tuples are themselves subject to eventual violation. For example, a nonstop flight from Los Angeles to Tel Aviv may be announced in the future. Thus, the database administrator may still need to verify that the assumptions hold.[2]

It is interesting to compare the integrity model presented here with the "traditional" approach to integrity. The accepted framework for defining integrity constraints under that approach is predicate calculus, and its main advantage is that integrity constraints may be monitored by the system itself. However, for reasons of efficient implementation, most systems allow only very simple predicates (e.g., referential constraints or range constraints). In this light, the following

[2] This is also true for "traditional" integrity constraints.

family of integrity constraints may be considered quite powerful, $(\forall x_1) \ldots (\forall x_n)(\alpha(x_1, \ldots, x_n) \Rightarrow \beta(x_1, \ldots, x_n))$, where x_i are domain variables and α and β are safe relational calculus expressions with these free variables. Such expressions may be rewritten as $\{(x_1, \ldots, x_n) \mid \alpha(x_1, \ldots, x_n) \wedge \neg\beta(x_1, \ldots, x_n)\} = \varnothing$. Therefore, these constraints are indeed statements of database views that must always be empty. In our model, these constraints are modeled as validity constraints that never include any true tuples. As discussed above, these constraints do not require human supervision either.

As an example, assume a system with the "traditional" integrity constraint "every flight from Los Angeles to Tel Aviv requires a stop-over." Before an update is accepted, the system will check whether this constraint would be violated. In our model this constraint will be represented as "nonstop flights from Los Angeles to Tel Aviv." Before an update is accepted the system will check whether it inserts a tuple into this view.

However, the new integrity model also allows validity constraints that *include* true tuples, requiring only that the false tuples that they include be excluded from the database. Thus, in this model it is possible to monitor the validity of views such as "nonstop flights from Los Angeles to New York." We conclude that the traditional integrity model (assuming the family of constraints defined above) is *subsumed* in the validity component of the new model.

4. THE INTEGRITY OF ANSWERS

In the previous sections we introduced integrity constraints and discussed how to enforce them in a dynamic environment, where both the database and the real world are changing. In addition to enforcing integrity, such constraints can be used for determining the integrity of answers issued by the database in response to queries. By accompanying its answers with statements of their integrity, the database system is, in effect, qualifying its answers by the scope of its knowledge. Such statements may be regarded as certification of the quality of the information delivered.

A *query* Q is an expression in the relation schemes of $\mathscr{R}$, that defines a new relation scheme, and for each database instance $\mathbf{D}$ defines a unique relation denoted $Q(\mathbf{D})$, which is the *answer* to the query in $\mathbf{D}$. We define the answer to a query Q to be *valid in* $\mathbf{D}$ (with respect to $\mathbf{W}$), if $Q(\mathbf{D}) \subseteq Q(\mathbf{W})$. That is, the answer from the database is contained in the answer from the real world. We define the answer to a query Q to be *complete in* $\mathbf{D}$ (with respect to $\mathbf{W}$), if $Q(\mathbf{W}) \subseteq Q(\mathbf{D})$. That is, the answer from the database contains the answer from the real world. An answer that is both valid and complete in $\mathbf{D}$ (with respect to $\mathbf{W}$) is said to have *integrity* in $\mathbf{D}$ (with respect to $\mathbf{W}$).

Of course, this definition of answer validity and completeness cannot be used to test whether the answer to a given query Q is valid or complete. Instead, we apply our knowledge about the views that are valid and complete. Intuitively, the answer to a query is valid or complete if it is "covered" by views that are known to be valid or complete.

Assume that a set of operations is available for manipulating views to create other views, while *preserving* the qualities of validity and completeness. Then, the sets $\mathscr{V}$ and $\mathscr{C}$ can be used to "span" additional views that are valid or

complete. If a given query Q can be derived from the views in $\mathscr{V}$, then its answer is valid. Similarly, if a given query Q can be derived from the views in $\mathscr{C}$, then its answer is complete.

Even when the answer to a query cannot be shown to be valid or complete, information about what *portions* of the answer are valid or complete is very useful.

Assume that V is a view of Q. If V is a validity constraint, then the answer to the query is guaranteed not to contain any false tuples from the view. We shall then say that Q is *valid with respect to* V. Similarly, if V is a completeness constraint, then the answer to the query is guaranteed to contain all the true tuples from the view. We shall then say that Q is *complete with respect to* V.

As an example, recall the validity constraint "nonstop flights from Los Angeles to New York" and the completeness constraint "flights from Los Angeles to New York via domestic carriers." The answer to the query "nonstop flights from Los Angeles to New York via TWA" is valid and complete (i.e., the answer has integrity). The answer to the query "flights from Los Angeles to New York that depart on Sundays" is valid only with respect to nonstop flights, and complete only with respect to domestic carriers (therefore, the answer has integrity only with respect to nonstop flights via domestic carriers).

Recall that if it is assumed that validity constraints never specify true tuples, then given a valid database instance **D**, there are no database tuples in the views of the validity constraints. Under this assumption, if Q is valid, its answer from **D** is guaranteed to be empty. We define such queries to be *unsatisfiable* in **D**. For example, consider a validity constraint "nonstop flights from Los Angeles to Tel Aviv," and the query "nonstop flights from Los Angeles to Tel Aviv that depart on Sunday." This query is unsatisfiable. Learning from a database system that a submitted query is unsatisfiable would be very valuable to the user; the null answer that would be delivered in such a case indicates only that *there are no data* that satisfy the query; the fact that a query is unsatisfiable indicates that *there may be no* data that satisfy the query.

5. REPRESENTING INTEGRITY CONSTRAINTS

In this and the following two sections we develop particular methods for expressing, storing, and manipulating integrity constraints, and show how they are used to determine the integrity of answers. For our examples we shall assume the database described in Figure 4.

Using domain relational calculus [9], a *view* is an expression of the form $\{(a_1, \ldots, a_n) \mid \psi(a_1, \ldots, a_n)\}$, where $a_1, \ldots, a_n$ are domain variables and ψ is a safe formula in predicate calculus with $a_1, \ldots, a_n$ as its only free variables. It defines a relation with the n-tuples of values that satisfy ψ.

Let $R_1, \ldots, R_n$ be the relation schemes. A *conjunctive* view [9] is an expression of the form:

$$\{(a_1, \ldots, a_n) \mid (\exists b_1) \cdots (\exists b_m)\ \psi_1 \wedge \cdots \wedge \psi_k\},$$

where the ψs may be of two kinds:

(1) *membership:* $(c_1, \ldots, c_p) \in R$, where R is a database relation scheme (of arity p), and the cs are either as or bs or constants.

CARRIER = (AIRLINE, NATIONALITY, FLEET)
= (FLIGHT-NO, AIRLINE, ORIGIN, DESTINATION, DEPART-DAY, FLIGHT STOPS)

Fig. 4. Scheme of database FLIGHTS.

(2) *comparative:* $d_1\ \theta\ d_2$, where d_1 is either an a or a b, d_2 is either an a or a b or a constant, and θ is a comparator (e.g., $<, \leq, >, \geq, =, \neq$).

In particular, each a and each b must appear at least once among the cs.

While conjunctive relational calculus expressions are a strict subset of the relational calculus, they are a powerful subset, corresponding to the set of relational algebra expressions with the operations *Cartesian product*, *selection*, and *projection* (where the selection predicates are conjunctive).

As an example, the view "stopover flights from Los Angeles to New York via domestic carriers" is expressed as follows:

$$\{(a) \mid (\exists b_1)(\exists b_2)(\exists b_3)(\exists b_4)(a, b_1, \text{LOS ANGELES}, \text{NEW YORK}, b_2, b_3) \in \text{FLIGHT} \wedge (b_1, \text{USA}, b_4) \in \text{CARRIER} \wedge b_3 > 0\}$$

For views from this family we have developed a representation that resembles regular data tuples. As we point out later, this approach provides important advantages. This method recalls the representation of QBE queries in skeleton tables [11].

For each relation R a *meta-relation* R' is defined. The scheme of R' is identical to the scheme of R, except for an additional attribute called IC. Also, an auxiliary relation is defined: COMPARISON = (X, COMPARE, Y). The meta-relations will be used to store membership subformulas of views. Their tuples will be referred to as *meta-tuples*. Comparative subformulas will be stored in relation COMPARISON.

Consider a view

$$\{(a_1, \ldots, a_n \mid (\exists b_1) \cdots (\exists b_m)\ \psi_1 \wedge \cdots \wedge \psi_k\},$$

A subformula ψ of the kind $(c_1, \ldots, c_p) \in R$ is first modified so that the cs that are as are suffixed with $*$, and the cs that are variables (i.e., as or bs) that appear only once in the whole expression are replaced with $\sqcup$ (blank). Hence, each component of the modified subformula is either a constant (a value), or a variable, or a blank, and each may be suffixed by $*$. This tuple is prefixed with either V or C, to indicate whether the corresponding view is a validity constraint or a completeness constraint, and is stored in R'. A subformula ψ of the kind $d_1 \theta d_2$, where θ is not $=$, is transformed to the tuple (d_1, θ, d_2) and stored in the auxiliary relation COMPARISON. If θ is $=$, then all occurrences of d_1 in the other subformulas are substituted with d_2. Finally, we assume that variable names are not shared among views.

As an example, consider the following six views:

$$\{(a_1, a_2) \mid (\exists b)(a_1, a_2, b) \in \text{CARRIER} \wedge a_2 = \text{USA}\} \tag{1}$$

$$\{(a_1, a_2, a_3) \mid (\exists b_1)(\exists b_2)(\exists b_3)(a_1, a_2, b_1, b_2, a_3, b_3) \in \text{FLIGHT} \wedge a_2 = \text{TWA} \wedge a_3 \neq \text{SATURDAY} \wedge a_3 \neq \text{SUNDAY}\} \tag{2}$$

$$\{(a_1, a_2, a_3, a_4, a_5) \mid (\exists b)(a_1, a_2, a_3, a_4, b, a_5) \in \text{FLIGHT} \wedge a_3 = \text{LOS ANGELES} \wedge a_4 = \text{NEW YORK} \wedge a_5 = 0\} \quad (3)$$

$$\{(a_1, a_2) \mid (\exists b)(a_1, b, a_2) \in \text{CARRIER} \wedge a_2 > 50\} \quad (4)$$

$$\{(a_1, a_2, a_3, a_4, a_5) \mid (\exists b_1)(\exists b_2)(\exists b_3)(a_3, a_1, a_4, a_5, b_2, b_3) \in \text{FLIGHT} \wedge (a_1, a_2, b_1) \in \text{CARRIER} \wedge a_2 = \text{USA} \wedge a_4 = \text{LOS ANGELES} \wedge a_5 = \text{NEW YORK}\} \quad (5)$$

$$\{(a_1, a_2, a_3) \mid (\exists b_1)(\exists b_2)(\exists b_3)(a_1, \text{USA}, b_1) \in \text{CARRIER} \wedge (a_2, a_1, \text{LOS ANGELES}, b_2, b_3, 0) \in \text{FLIGHT} \wedge (a_3, a_1, b_2, \text{NEW YORK}, b_3, 0) \in \text{FLIGHT}\} \quad (6)$$

Assume that the first three views are validity constraints and the other three views are completeness constraints. Figure 5 shows a small instance of the database together with these six constraints. For convenience of presentation, each pair of relations, R, R' is shown as a single contiguous table, and the attribute IC is also used to identify the meta-tuples. The first constraint ("domestic carriers") is represented with the meta-tuple V1. The second constraint ("mid-week flights via TWA") is represented with the meta-tuple V2 and the first two tuples in COMPARISON. The third constraint ("nonstop flights from Los Angeles to New York") is represented with the meta-tuple V3. The fourth constraint ("large carriers") is represented with the meta-tuple C1 and the third tuple in COMPARISON. The fifth constraint ("flights from Los Angeles to New York via domestic carriers") is represented with the meta-tuples C2 and C3. The sixth constraint ("pairs of nonstop flights via the same domestic carrier that depart on the same day, where the first flight originates in Los Angeles, and the second flight departs from the destination of the first flight and arrives at New York") is represented with the meta-tuples C4, C5, and C6. Note that many of the views are defined to include the selection attributes in the projection attributes. The advantage of such views will be evident later.

Indeed, each individual meta-tuple stored in relation R' may be regarded as defining a view of R: The constants and variables specify the selection condition, and the *s specify the projected attributes. For example, the meta-tuple V1 specifies a selection of all tuples of relation CARRIER for which NATIONALITY = USA and a projection on AIRLINE and NATIONALITY. (Note that a variable shared by another meta-tuple, such as x_3 in the meta-tuple C2, specifies a selection condition which is satisfied by any value from a set of values defined elsewhere.) We observe that each individual meta-tuple expresses an integrity constraint that is satisfied by every instance of the database that satisfies the constraint to which it belongs (the view defined by each meta-tuple is simply the projection of the entire view on the attributes in which the meta-tuple has *s).

This method for storing constraints has several advantages. First, the specification of integrity constraints using QBE-like notation is very intuitive. Second, storing the constraints does not require any new data structures. Third, the constraints may be updated with tools similar to those used to update the data. Finally, this representation allows us to develop a method that determines the integrity of an answer, in a process analogous to the determination of the answer itself.

CARRIER

IC	AIRLINE	NATIONALITY	FLEET
	AIR FRANCE	FRANCE	63
	AMERICAN	USA	156
	EL AL	ISRAEL	18
			52
	JAL	JAPAN	44
	BA	BRITAIN	195
	TWA	USA	230
	UNITED	USA	42
	VARIG	BRAZIL	
V1	*	USA*	
C1	*		x_2*
C2	x_3*	USA*	
C4	x_4*	USA	

COMPARISON

X	θ	Y
x_1	$\neq$	SATURDAY
x_1	$\neq$	SUNDAY
x_2	$>$	50

FLIGHT

IC	FLIGHT-NO	AIRLINE	ORIGIN	DESTINATION	DEPART-DAY	STOPS
	AA-360	AMERICAN	LOS ANGELES	PHILADELPHIA	SUNDAY	0
	JA-400	JAL	NEW YORK	TOKYO	WEDNESDAY	2
	BA-021	BA	BOSTON	LONDON	MONDAY	1
	LY-006	EL AL	LOS ANGELES	TEL AVIV	WEDNESDAY	1
	TW-391	TWA	LOS ANGELES	NEW YORK	SUNDAY	0
	UA-366	UNITED	LOS ANGELES	NEW YORK	SUNDAY	1
	VG-183	VARIG	RIO DE JANEIRO	CHICAGO	FRIDAY	2
V2	*	TWA*			x_1*	
V3	*	*	LOS ANGELES*	NEW YORK*		0*
C3	*	x_3*	LOS ANGELES*	NEW YORK*		
C5	*	x_4*	LOS ANGELES	x_5	x_6	0
C6	*	x_4*	x_5	NEW YORK	x_6	0

Fig. 5. A database extended with constraints.

6. DETERMINING THE INTEGRITY OF ANSWERS

Consider a database scheme with relation schemes $R_1, \ldots, R_n$, and meta-relation schemes $R_1, \ldots, R_n'$ and COMPARISON. Assume that the meta-relations include definitions for validity constraints $V_1, \ldots, V_k$ and completeness constraints $C_1, \ldots, C_m$.

Assume a set of view operations that preserve validity and completeness, and consider a query Q submitted against an instance of this database. If Q is a view of the validity constraints $V_1, \ldots, V_k$, then its answer is valid; if Q is a view of the completeness constraints $C_1, \ldots, C_m$, then its answer is complete.

Also, if any view of Q is a view of $V_1, \ldots, V_k$, then the answer is valid with respect to this view; if any view of Q is a view of $C_1, \ldots, C_m$, then the answer is complete with respect to this view.

We describe a method that discovers views of a given query that are valid or complete. Basically, this method *generates* valid or complete views of Q, by manipulating the definitions of the constraints algebraically. These manipula-

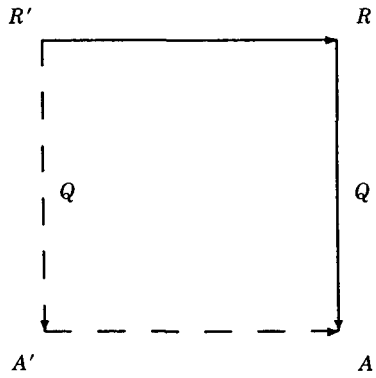

Fig. 6. Extending query processing to manipulate constraints.

tions mirror those that are necessary to implement Q. In effect, we *generalize* the standard Cartesian product, selection and projection operations to manipulate also meta-relations of view definitions.

This method is illustrated by the commutative diagram shown in Figure 6. The solid lines describe the current situation: the meta-relations R' define the validity and completeness of the database relations R, and the virtual relation A is derived from R to answer query Q. The dashed lines describe our method: query processing is extended to manipulate also R', to yield the meta-relation A' that defines the validity and completeness of the answer A.

We shall assume queries are from the same family of conjunctive relational calculus expressions. Recall that these are also the queries that can be expressed with the algebraic operations Cartesian product, selection, and projection. For simplicity, in the remainder of this section we shall ignore the attribute IC in the meta-relations.

6.1 Meta-Relation Operations

Definition 1. Assume that R' and S' are meta-relations that define integrity constraints on R and S. The Cartesian product of R' and S', denoted $R' \times S'$, is defined as follows. For every pair r and s of validity (completeness) meta-tuples from R' and S', respectively,

$$r = (r_1, \ldots, r_m)$$
$$s = (s_1, \ldots, s_n),$$

$R' \times S'$ includes the validity (completeness) metatuple

$$q = (r_1, \ldots, r_m, s_1, \ldots, s_n).$$

PROPOSITION 1. *Let r, s, and q be as in Definition* 1, *let* **D** *be an instance of this database, and let $r(\mathbf{D})$, $s(\mathbf{D})$, and $q(\mathbf{D})$ denote, respectively, the relations defined by r, s, and q. Then $q(\mathbf{D}) = r(\mathbf{D}) \times s(\mathbf{D})$.*

PROOF. Let λ and μ denote, respectively, the selection predicates of r and s, and let α and β denote, respectively, the projected attributes of r and s. Then

$r(\mathbf{D})$, $s(\mathbf{D})$ and $q(\mathbf{D})$ can be expressed as the following Cartesian product-selection-projection expressions:

$$r(\mathbf{D}) = \pi_{\alpha}\sigma_{\lambda}(R(\mathbf{D}))$$
$$s(\mathbf{D}) = \pi_{\beta}\sigma_{\mu}(S(\mathbf{D}))$$
$$q(\mathbf{D}) = \pi_{\alpha\cup\beta}\sigma_{\lambda\wedge\mu}(R(\mathbf{D}) \times S(\mathbf{D}))$$

We observe that $\pi_{\alpha\cup\beta}\sigma_{\lambda\wedge\mu}(R(\mathbf{D}) \times S(\mathbf{D})) = \pi_{\alpha}\sigma_{\lambda}(R(\mathbf{D})) \times \pi_{\beta}\sigma_{\mu}(S(\mathbf{D}))$. □

PROPOSITION 2. *The constraints in $R' \times S'$ are satisfied by every database instance that satisfies the constraints in R' and S'.*

PROOF. Assume first that r and s are validity constraints. Let $(a_1, \ldots, a_k)$ be a tuple from $q(\mathbf{D})$. Since $q(\mathbf{D}) = r(\mathbf{D}) \times s(\mathbf{D})$, there is an integer j such that $(a_1, \ldots, a_j)$ is a tuple in $r(\mathbf{D})$ and $(a_{j+1}, \ldots, a_k)$ is a tuple in $s(\mathbf{D})$. Since $r(\mathbf{D})$ and $s(\mathbf{D})$ are valid, $(a_1, \ldots, a_j)$ is also in $r(\mathbf{W})$ and $(a_{j+1}, \ldots, a_k)$ is also in $s(\mathbf{W})$. Since $q(\mathbf{W}) = r(\mathbf{W}) \times s(\mathbf{W})$, $(a_1, \ldots, a_j, a_{j+1}, \ldots, a_k)$ is a tuple in $q(\mathbf{W})$. Hence q is valid in $\mathbf{D}$. Similarly, if r and s are completeness constraints. □

Definition 2. Assume that R' is a meta-relation that defines integrity constraints on R. Let λ denote a primitive selection predicate (i.e., either $R_i'\theta c$, or $R_i'\theta R_j'$). The selection from R' by predicate λ, denoted $\sigma_{\lambda}(R')$, is defined as follows. Consider first the case $\lambda = R_i\theta c$, and let r be a validity (completeness) meta-tuple from R',

$$r = (r_1, \ldots, r_i, \ldots, r_m).$$

Denote by μ the selection predicate expressed by r_i[3]. If r_i is suffixed by $*$, then $\sigma_{\lambda}(R')$ includes the validity (completeness) meta-tuple:

$$q = (r_1, \ldots, r_i', \ldots, r_m),$$

where r_i' represents $\lambda \wedge \mu$. Consider now the case $\lambda = R_i\theta R_j$, and let r be a validity (completeness) meta-tuple from R',

$$r = (r_1, \ldots, r_i, \ldots, r_j, \ldots, r_m).$$

Denote by μ the selection predicate expressed by r_i and r_j. If r_i and r_j are both suffixed by $*$, then $\sigma_{\lambda}(R')$ includes the validity (completeness) meta-tuple:

$$q = (r_1, \ldots, r_i', \ldots, r_j', \ldots, r_m),$$

where r_i' and r_j' represent $\lambda \wedge \mu$.

PROPOSITION 3. *Let r and q be as in Definition 2, let $\mathbf{D}$ be an instance of this database, and let $r(\mathbf{D})$ and $q(\mathbf{D})$ denote, respectively, the relations defined by r and q. Then $q(\mathbf{D}) = \sigma_{\lambda}r(\mathbf{D})$.*

PROOF. Let α denote the projected attributes of r. Then $r(\mathbf{D})$ and $q(\mathbf{D})$ can be expressed as the following selection-projection expressions:

$$r(\mathbf{D}) = \pi_{\alpha}\sigma_{\mu}(R(\mathbf{D}))$$
$$q(\mathbf{D}) = \pi_{\alpha}\sigma_{\mu\wedge\lambda}(R(\mathbf{D})).$$ □

[3] If r_i is blank, then μ is true.

We observe that if the predicate λ is on attributes in α, then $\pi_\alpha\sigma_{\mu\wedge\lambda}(R(\mathbf{D})) = \sigma_\lambda\pi_\alpha\sigma_\mu(R(\mathbf{D}))$.

PROPOSITION 4. *The constraints in $\sigma_\lambda(R')$ are satisfied by every instance of this database that satisfies the constraints in R'.*

PROOF. Assume first that r is a validity constraint. Let $(a_1, \ldots, a_k)$ be a tuple from $q(\mathbf{D})$. Then, since $q(\mathbf{D}) = \sigma_\lambda r(\mathbf{D})$, $(a_1, \ldots, a_k)$ is also in $r(\mathbf{D})$ and it satisfies λ. Since $r(\mathbf{D})$ is valid, $(a_1, \ldots, a_k)$ is also in $r(\mathbf{W})$. Since it satisfies λ and $q(\mathbf{W}) = \sigma_\lambda r(\mathbf{W})$, $(a_1, \ldots, a_k)$ is also in $q(\mathbf{W})$. Hence, q is valid in $\mathbf{D}$; similarly, if r is a completeness constraint. □

Definition 3. Assume that R' is a meta-relation that defines integrity constraints on R. The projection of R' that removes its i'th attribute, denoted $\pi_{R-R_i}(R')$, is defined as follows. For every validity (completeness) meta-tuple r from R',

$$r = (r_1, \ldots, r_m),$$

if r_i is ⊔ (possibly suffixed with $*$), then $\pi_{R-R_i}(R')$ includes the validity (completeness) meta-tuple:

$$q = (r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_m).$$

PROPOSITION 5. *Let r and q be as in Definition* 3, *let* $\mathbf{D}$ *be an instance of this database, and let $r(\mathbf{D})$ and $q(\mathbf{D})$ denote, respectively, the relations defined by the meta-tuples r and q. Then $q(\mathbf{D}) = \pi_{R-R_i}(r(\mathbf{D}))$.*[4]

PROOF. Let λ denote the selection predicate of r, let α denote the projected attributes of r, and let $\beta = R - R_i$. Then $r(\mathbf{D})$ and $q(\mathbf{D})$ can be expressed as the following selection-projection expressions:

$$r(\mathbf{D}) = \pi_\alpha\sigma_\lambda(R(\mathbf{D}))$$
$$q(\mathbf{D}) = \pi_\alpha\sigma_\lambda\pi_\beta(R(\mathbf{D})).$$

We observe that if the ith attribute of R does not participate in the predicate λ, then $\pi_\alpha\sigma_\lambda\pi_\beta(R(\mathbf{D})) = \pi_\beta\pi_\alpha\sigma_\lambda(R(\mathbf{D}))$.

PROPOSITION 6. *The constraints in $\pi_{R-R_i}(R')$ are satisfied by every instance of this database that satisfies the constraints in R'.*

PROOF. Assume first that r is a validity constraint. Let $(a_1, \ldots, a_k)$ be a tuple from $q(\mathbf{D})$. Assume first that R_i is removed by r. Then, since $q(\mathbf{D}) = \pi_{R-R_i}(r(\mathbf{D}))$, $(a_1, \ldots, a_k)$ is also in $r(\mathbf{D})$. Since $r(\mathbf{D})$ is valid, $(a_1, \ldots, a_k)$ is also in $r(\mathbf{W})$. Since $q(\mathbf{W}) = \pi_{R-R_i}(r(\mathbf{W}))$, $(a_1, \ldots, a_k)$ is also in $q(\mathbf{W})$. Assume now that R_i is not removed by r. Then, since $q(\mathbf{D}) = \pi_{R-R_i}(r(\mathbf{D}))$, there is a constant a' and an integer j, such that $(a_1, \ldots, a_j, a', a_{j+1}, \ldots, a_k)$ is a tuple in $r(\mathbf{D})$. Since $r(\mathbf{D})$ is valid, $(a_1, \ldots, a_j, a', a_{j+1}, \ldots, a_k)$ is also in $r(\mathbf{W})$. Since $q(\mathbf{W}) = \pi_{R-R_i}(r(\mathbf{W}))$, $(a_1, \ldots, a_j, a_{j+1}, \ldots, a_k)$ is a tuple in $q(\mathbf{W})$. Hence q is valid in $\mathbf{D}$. Similarly, if r is a completeness constraint.

[4] In general, we define $\pi_\alpha(R)$ as a projection on those attributes in α that are in R. Thus, if attribute R_i had already been removed, a projection on $R - R_i$ has no effect.

Briefly, the selection and projection operations behave as follows: selection requires the attributes it selects to be in the projection attributes of the meta-tuple; and projection requires the attribute it removes not to be in the selection attributes of the meta-tuple. This behavior can also be justified intuitively, as follows.

Consider first the selection operation. Assume a database scheme with one relation scheme R = (A, B), and let R(**D**) and R(**W**) be as follows:

R(**D**)

A	B
u	1
x	5
y	6

R(**W**)

A	B
x	2
y	3
z	4

Assume the validity constraint $v = (*, b)$, where $b \geq 2$. **D** satisfies v, because $v(\mathbf{D}) = \{x, y\}$, while $v(\mathbf{W}) = \{x, y, z\}$. However, the constraint $v' = (*, b)$, where $b \geq 4$, is not valid, because $v'(\mathbf{D}) = \{x, y\}$, while $v'(\mathbf{W}) = \{z\}$. Hence, it is not true that any selection from a valid view generates a valid view. Even though the selection $b \geq 2$ defined a valid view, further selection did not retain this validity. This is because the validity of v did not cover the values of attribute B!

Next, consider the projection operation. Assume the same database scheme with the following instances:

R(**D**)

A	B
u	1
x	2
y	3

R(**W**)

A	B
x	2
y	3
z	4

Assume the same validity constraint $v = (*, b)$, where $b \geq 2$. **D** satisfies v, because $v(\mathbf{D}) = \{x, y\}$, while $v(\mathbf{W}) = \{x, y, z\}$. While it is true that every projection of a valid view generates a valid view, if the projection removes attribute B, it is impossible to express the new view with the remaining attributes!

Propositions 2, 4 and 6 guarantee that the three algebraic operations for meta-relations generate views that "preserve" validity and completeness. More accurately, when the same algebraic expression in these operations is applied to a set of relations and to their corresponding meta-relations, the resulting meta-relation expresses constraints on the resulting relation that are satisfied by every database instance that satisfies the constraints expressed in the meta-relations. Noting that any conjunctive query may be implemented with Cartesian products of two relations, selections by a primitive predicate, and projections that remove a single attribute, the foregoing results are stated formally in the following theorem.

THEOREM. *Assume a database scheme $\mathscr{R}$, validity constraints $\mathscr{V}$, and completeness constraints $\mathscr{C}$. Let Q be a conjunctive query against this database. Let S be the relational algebra expression that implements Q. Let S' be the relational algebra*

expression obtained from S by substituting every reference to R with a reference to R′. S operates on the actual database relations to yield the answer A. S′ operates on the meta-relations to yield the meta-relation A′ of constraints on A. Then, the constraints in A′ are satisfied by every database instance that satisfies the constraints in $\mathscr{V}$ and $\mathscr{C}$.

The theorem guarantees that meta-tuples in A' are views of A that are either valid or complete. However, some meta-tuples may still contain references to meta-tuples outside A', and are therefore not expressible entirely within A'. Such views are avoided if S' is modified so that all Cartesian products are performed first, and their result is pruned to retain only those meta-tuples that do not contain references to other meta-tuples. Also, as we explain below, it is advantageous to perform selections before projections. Altogether, S' is transformed to a sequence of Cartesian products, followed by selections, and ending with projections. This simple strategy for implementing conjunctive queries is not necessarily optimal. However, we note that the optimality is not so essential for meta-relations, because they are relatively small. For the actual relations, where optimality is essential, a different strategy may be implemented.

6.2 Refinements

The theorem guarantees that the method for generating integrity constraints is *sound*, but it does not guarantee that it is *complete*. That is, this method generates views of the result that indeed have integrity, but does not necessarily generate all such views.[5] A method that would guarantee completeness would undoubtedly be of a different complexity altogether. However, for our purpose here, to improve the quality of database answers, completeness is not an absolute requirement. Yet, with several simple refinements, it can be improved to generate additional desirable views. Three such refinements are sketched below.

When an attribute of R' is removed, all the meta-tuples that restrict this attribute (with a variable or a constant) are discarded. Consequently, some views may be lost. For example, assume that Q is a Cartesian product of R and S, followed by a projection that removes all the attributes of S. Obviously, Q is equivalent to R, and A' should retain all the meta-tuples of R'. However, these meta-tuples may be discarded by the projection, if they contain restrictions in the attributes contributed by S'. To handle this situation we may extend the Cartesian product of meta-relations to include also these two tuples:

$$q_1 = (r_1, \ldots, r_m, \sqcup, \ldots, \sqcup)$$
$$q_2 = (\sqcup, \ldots, \sqcup, s_1, \ldots, s_n).$$

These tuples define all previous constraints on R and S as constraints on the Cartesian product of R and S.

During selection, certain constraints may be *cleared* from the selected meta-tuple (e.g., a variable or a constant is replaced by $\sqcup$). If the selection predicate λ *implies* the constraint predicate μ (i.e., the view is broader than the query), then considering the semantics of the resulting relation, the new view should not

[5] This *logical* completeness should not be confused with *database* completeness, which is the topic of this article.

restrict this attribute any more. For example, recall the meta-tuple V3, which asserts the validity of the view "nonstop flights from Los Angeles to New York." Consider now the selection $\sigma_{\text{ORIGIN}=\text{LOS ANGELES}}(\text{FLIGHT})$. The semantics of the resulting relation are no longer "flights" but "flights originating in Los Angeles." Hence, on this new relation, the view "nonstop flights to New York" is valid. Clearing selection constraints ensures that more meta-tuples will "survive" future projections.

Clearly, the selection operation requires the most effort. The operation itself requires conjoining μ, the predicate expressed in the meta-tuple, with λ, the predicate expressed in the query. And the above refinement is possible only if it can be determined that λ implies μ. (Of course, if it is determined first that λ implies μ, then μ is cleared and it is not necessary to generate their conjunction.) Often, these tasks are quite simple, as in the above example, where λ and μ are identical. At other times it may require consulting relation COMPARISON, and possibly modifying it. Particular implementations may choose to perform these tasks only for certain types of λ and μ. In such cases, when other types are involved, the relevant meta-tuple must not be selected. Note that the only other time when relation COMPARISON is used is when the views in A' are described to the user.

Finally, with a simple procedure, it is often possible to infer additional constraints from constraints that are stored in the same meta-relation. For example, assume that CARRIER′ has two meta-tuples, (*, *, ⊔) and (*, ⊔, *). A query that selects *both* NATIONALITY and FLEET will not select any of these meta-tuples. However, it can be shown that in this case (*, *, *) is also a constraint on CARRIER, and this meta-tuple would have been selected by the same query. Let r and s be meta-tuples in relation R', such that either both belong to validity constraints or both belong to completeness constraints, but both do not belong to the same constraint. Assume that the views defined by r and s can participate in a *lossless join* (for example, both views include the key of this relation). We define their *self-join* with a meta-tuple q, as follows: q_i is the disjunction of the constraints defined in r_i and s_i, and it is suffixed by * if both r_i or s_i are suffixed by *. It can be shown that self-joins are also constraints on R. Note that self-joins need not be generated for every query; once generated, they should be stored with the original view definitions, until these definitions are modified.

7. EXAMPLES

As a first example, consider the database of Figure 5 and the query "details of flights via domestic carriers originating in Los Angeles." For clarity, this query will be processed in five steps:

(1) Perform the Cartesian product of FLIGHT and CARRIER.
(2) Select from the result FLIGHT.AIRLINE = CARRIER.AIRLINE.
(3) Remove from the result one of the attributes AIRLINE.
(4) Select from the result ORIGIN = LOS ANGELES and NATIONALITY = USA.
(5) Remove from the result attributes ORIGIN, NATIONALITY and FLEET.

The first three steps correspond to a *natural join* between the two relations. Their result is shown in Table I.

Table I

IC	FL-NO	AIRLINE	ORIGIN	DESTINATION	DEP-DAY	ST	NATION	FLT
	AA-360	AMERICAN	LOS ANGELES	PHILADELPHIA	SUNDAY	0	USA	156
	JA-400	JAL	NEW YORK	TOKYO	WEDNESDAY	2	JAPAN	52
	BA-021	BA	BOSTON	LONDON	MONDAY	1	BRITAIN	44
	LY-006	EL AL	LOS ANGELES	TEL AVIV	WEDNESDAY	1	ISRAEL	18
	TW-391	TWA	LOS ANGELES	NEW YORK	SUNDAY	0	USA	195
	UA-366	UNITED	LOS ANGELES	NEW YORK	SUNDAY	1	USA	230
	VG-183	VARIG	RIO DE JANEIRO	CHICAGO	FRIDAY	2	BRAZIL	42
V4	*	TWA*			x_1*		USA*	
V5	*	*	LOS ANGELES*	New York*		0*	USA*	
C7	*	*	LOS ANGELES*	New York*			USA*	

Table II

IC	FL-NO	AIRLINE	ORIGIN	DESTINATION	DEP-DAY	ST	NATION	FLT
	AA-360	AMERICAN	LOS ANGELES	PHILADELPHIA	SUNDAY	0	USA	156
	TW-391	TWA	LOS ANGELES	NEW YORK	SUNDAY	0	USA	195
	UA-366	UNITED	LOS ANGELES	NEW YORK	SUNDAY	1	USA	230
V6	*	*	*	NEW YORK*		0*	*	
C8	*	*	*	NEW YORK*			*	

Table III

IC	FLIGHT-NO	AIRLINE	DESTINATION	DEPART-DAY	STOPS
	AA-360	AMERICAN	PHILADELPHIA	SUNDAY	0
	TW-391	TWA	NEW YORK	SUNDAY	0
	UA-366	UNITED	NEW YORK	SUNDAY	1
V7	*	*	NEW YORK*		0*
C9	*	*	NEW YORK*		

V4 was obtained from V1 and V2; V5 was obtained from V1 and V3; and C7 was obtained from C2 and C3. The fourth step now selects V5 and C7 (clearing the appropriate fields). Its result is shown in Table II.

The projections retain both meta-tuples, and the final result is shown in Table III.

These are the details of flights via domestic carriers originating in Los Angeles. The meta-tuples indicate that the nonstop flights to New York (flight number and airline) are guaranteed to be valid, and that all flights to New York (flight number and airline) are included. Recalling the original six views, the validity view in the final derivation is a view of the Cartesian product of the first and third views, and the completeness view in the final derivation is a view of the fifth view.

For a second example, consider the database of Figure 5, with one additional constraint: There are no El Al flights that depart on Saturday. This constraint may be specified with the following validity constraint that never includes true

Table IV

IC	FLIGHT-NO	DESTINATION
I2	*	*

tuples: "Flights via El Al that depart on Saturday." Such validity constraints should be handled separately from other validity constraints, and we shall use the indicator I. An appropriate meta-tuple in relation FLIGHT would be: (I1, *, EL AL*, *, *, SATURDAY*, *).

Consider now the query "flight number and destination of flights via El Al originating in Los Angeles that depart on Saturday." The processing of this query yields the relation shown in Table IV.

As there are no Saturday flights via El Al from Los Angeles, the result includes no data tuples. However, the meta-tuple indicates that the query is indeed unsatisfiable.

8. CONCLUSION

Current database systems cannot fully guarantee the integrity of the information they deliver in response to queries. Only a limited kind of integrity is guaranteed, and the user has no way of knowing precisely which portions of an answer are "covered" under this guarantee and which are not.

To correct these limitations, this article presented a new model for integrity of relational databases. The model establishes new criteria for database integrity, called validity and completeness. It introduces new kinds of integrity constraints to express validity and completeness of specific views, and it defines the validity and completeness of answers issued by the database. The issue of maintaining integrity in a dynamic environment was considered, and a particular kind of integrity constraint that does not require human supervision was isolated. The new model can represent a large subset of the traditional predicate calculus constraints.

A specific mechanism was described that implements the principles of this model. A simple, yet powerful, family of integrity constraints was isolated. For this family, a representation was developed that stores the constraints in meta-relations that mirror the actual relations. Relational algebra was then extended to manipulate these meta-relations, so that the meta tuples in a resulting meta-relation assert correctly the integrity of the data tuples in the resulting relation. The main advantage of this representation is that the standard query evaluation process yields, almost as a by-product, the information on the integrity or partial integrity of the answer.

A prototype implementation of this mechanism is currently being developed. This prototype is implemented as a pair of external interfaces to the database system INGRES. One interface allows the database administrator to define and modify integrity constraints. The other interface solicits QUEL queries from the user, performs them on the meta-relations, and, in parallel, submits them to the database system. The answer obtained from the database system is presented to

the user, along with a QUEL representation of its integrity, as obtained from the meta-relations.

The new integrity model and the specific mechanism may be viewed as independent research results. The mechanism offers a specific solution to the following general problem (described in Section 6): Given a set of views $\mathscr{V}$ and a view V, what are the views of V that are views of $\mathscr{V}$? (and, in particular, is V itself a view of $\mathscr{V}$?). This problem appears in several other database areas, and the mechanism described here can be applied to solve these problems, as well. For example, given a set of views that a user is *authorized to access*, should a particular query submitted by this user be authorized (and, if not, what portions of the answer may be authorized?). On the other hand, the benefits of the new integrity model may be realized with an altogether different mechanism.

There are several issues that require further investigation, and we mention here three examples.

The specific integrity mechanism is simple and efficient, but has two limitations. First, views and queries are currently limited to conjunctive formulas. Here, we are investigating several extensions that will broaden the type of views that can be handled; for example, allowing formulas with disjunctions or formulas with aggregate functions. Second, the method for determining the integrity of answers is not necessarily complete. Here also, we are investigating several extensions that will generate additional definitions of views with integrity; for example, by simply avoiding the final projections in S', it should be possible to retain the definitions of those views of the result that are expressed with attributes that eventually are removed from the result.

In Section 3 we isolated a particular kind of integrity constraint that, like traditional integrity constraints, can be enforced without human supervision. How to enforce them *efficiently* is an issue that still needs to be investigated. For example, it should be quite simple to detect whether a tuple that is about to be added to a relation would violate any constraint that is expressed entirely within this relation (i.e., without joins).

In general, the responsibility for enforcing integrity constraints lies with the database administrator, and all integrity certifications that accompany answers are founded on the assumption that all constraints are presently satisfied. In a multi-user environment, it may be desirable to allow different users to contribute validity and completeness constraints. It is possible to extend the model and the mechanism to associate with each constraint the name of the user who contributed it. This may be done with another pseudo-attribute to be attached to meta-tuples (like IC). Certifications would then be annotated with the name of the user who contributed the constraints involved (e.g., "according to Smith, the following view is valid . . ."). Similarly, it is possible that some integrity constraints are not presently satisfied, because the database administrator (or the contributing user) can only verify them periodically. It is possible to extend the model and the mechanism to associate with each constraint the time at which it was last verified. Again, this may be done with a pseudo-attribute. Certifications would then be annotated with the time at which the constraints involved were last verified (e.g., "as of 10-Dec-87 10:22, the following view is complete . . ."). An interesting issue here is how to combine views that were contributed by different users and verified at different times.

ACKNOWLEDGMENTS

The author is grateful to the Associate Editor and the reviewers for their helpful suggestions.

REFERENCES

1. DATE, C. J. *An Introduction to Database Systems, Volume I.* 4th Ed. Addison Wesley, Reading, Mass., 1986.
2. KORTH, H. F., AND SILBERSCHATZ, A. *Database System Concepts.* McGraw-Hill, New York, 1986.
3. LEVESQUE, H. J. The logic of incomplete databases. In *On Conceptual Modelling: Perspectives from Artificial Intelligence Databases and Programming Languages*, J. Mylopoulos, M. L. Brodie, and J. W. Schmidt, Eds., Springer-Verlag, Berlin, 1984, pp. 165–186.
4. MAIER, D. *The Theory of Relational Databases.* Computer Science Press, Rockville, Md., 1983.
5. MOTRO, A. Completeness information and its application to query processing. In *Proceedings of the 12th International Conference on Very Large Data Bases* (Kyoto, Aug. 25–28, 1986), pp. 170–178, VLDB Endowment (available from Morgan-Kaufmann, Los Altos, Calif., 1986).
6. MOTRO, A. SEAVE: A mechanism for verifying user presuppositions in query systems. *ACM Trans. Off. Inf. Syst. 4*, 4 (Oct. 1986), 312–330.
7. REITER, R. On closed world data bases. In *Logic and Databases*, H. Gallaire and J. Minker, Eds., Plenum Press, New York 1978, pp. 55–76.
8. *SunINGRES Manual Set.* Release 5.0, Sun Microsystems, Mountain View, Calif., (Part Number 800-1644-01), 1987.
9. ULLMAN, J. D. *Principles of Database Systems.* Computer Science Press, Rockville, Md., 1982.
10. *UNIFY Reference Manual.* 3.0 edition, UNIFY Corp., Lake Oswego, Oreg., 1983.
11. ZLOOF, M. Query-by-Example: A database language. *IBM Syst. J. 16*, 4 (Dec. 1977), 324–343.

Received August 1986; revised December 1987 and January 1989; accepted April 1989