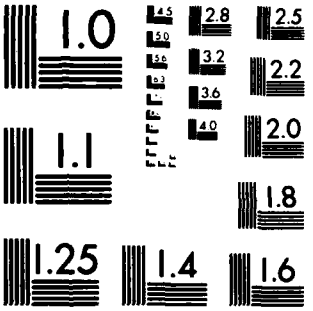
AD-A144 319 APPROACHES FOR UPDATING DATABASES WITH INCOMPLETED INFORMATION AND NULLS(U) STANFORD UNIV CA DEPT OF COMPUTER SCIENCE A KELLER ET AL. APR 84 1/1

UNCLASSIFIED AFOSR-TR-84-0382 N00039-82-G-0250 F/G 9/2 NL



UNCLASSIFIED
SECURITY CLASSIFICATION

AD-A144 319

(2)

DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 84-0582
6a. NAME OF PERFORMING ORGANIZATION Stanford University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research
6c. ADDRESS (City, State and ZIP Code) Computer Science Department Stanford CA 94305	7b. ADDRESS (City, State and ZIP Code) Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR 8b. OFFICE SYMBOL (If applicable) NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR-80-0212
8c. ADDRESS (City, State and ZIP Code) Bolling AFB DC 20332	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. A2; WORK UNIT NO.
11. TITLE (Include Security Classification) APPROACHES FOR UPDATING DATABASES WITH INCOMPLETED INFORMATION AND NULLS	
12. PERSONAL AUTHOR(S) Arthur Keller and Marianne Winslett Wilkins	
13a. TYPE OF REPORT Reprint 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) April 1984 15. PAGE COUNT 9
16. SUPPLEMENTARY NOTATION IEEE Computer Data Engineering Conference Proceedings, April 1984, Los Angeles CA.	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this paper we consider approaches to updating databases containing null values and incomplete information. Our approach distinguishes between modeling incompletely known worlds and modeling changes in these worlds. As an alternative to the open and closed world assumptions, they propose the modified closed world assumption. Along with the discussion of updating, they address some issues of refining incompletely specified information.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☑ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Robert N. Buchal	22b. TELEPHONE NUMBER (Include Area Code) (202) 767-4939 22c. OFFICE SYMBOL NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

IEEE 3S CONFERENCE
Computer Data
Engineering Conf.
April 1984
Los Angeles

Approaches for Updating Databases With Incomplete Information and Nulls

Arthur Keller and Marianne Winslett Wilkins
Stanford University, Computer Science Dept.

Abstract. In this paper we consider approaches to updating databases containing null values and incomplete information. Our approach distinguishes between modeling incompletely known worlds and modeling changes in these worlds. As an alternative to the open and closed world assumptions, we propose the modified closed world assumption. Along with the discussion of updating, we address some issues of refining incompletely specified information.

Key Words and Phrases. Null values, incomplete information, updates, databases, relational databases.

CR Categories. H.2.1, H.2.3, H.1.1.

1. Introduction

The real world, and a database that models it, varies over time. At each moment in time, we have a world state and a corresponding database state. As the world state changes with time, we want the database to track these changes. We distinguish between these two problems, that of modeling an incompletely known world and of modeling changes in that world. In the sections below, we first discuss modeling incompletely known *static* worlds, with updates serving the purpose of refining the database when more complete information is known. Following this discussion, we address some issues concerning the use of updates to handle changes in the world state.

1a. Why do we have incomplete information about the real world?

Incomplete information arises from several sources. At first, in the initial stages of using a new system, not all of the necessary information may have been captured, resulting in an incompletely known static world state. Later on, new information may become available in a piecemeal fashion, resulting in an incompletely specified change to a world state. In addition, for privacy or security reasons we may not want to store particular information for certain members of a domain, giving us an incompletely known static world state. Some information may be omitted because it is quite expensive or difficult to obtain. In a shared database, the responsibility for capturing the information may be decentralized. Users' views may omit information stored in the database [Chamberlin 75, Stonebraker 75]. Consequently, view updates [Dayal 82, Keller 82] often result in incomplete information. Finally, some attributes may be inapplicable in a particular situation, indicating that the structure of the database model does not exactly correspond to the structure of the world.

1b. Alternative Worlds

Given an incomplete body of knowledge about a world, we expect to find multiple worlds satisfying that body of knowledge. If we consider the body of knowledge to be a theory, then the possible worlds are models that satisfy that theory.

We may choose to apply constraints to the relationship between these models and the original theory. One such constraint, known as the *open world assumption*, states that the theory is correct but not necessarily complete. That is, if the negation of a fact can be derived from the theory, then that fact must be false in all models. There can be facts true in some models (and false in others) that are not conclusively specified in the theory. This gives us three classes of statements: those true in all models, those false in all models, and those true in some models and false in others (hereafter referred to as "true," "false," and "maybe" statements or results, respectively). We shall use the term *definite results* to refer to the "true" and "false" results.

Another constraint, the *closed world assumption* [Reiter 78, 80], states that all relevant information is given in the database. That is, if a fact cannot be derived from the theory, its negation may be assumed to hold. A database is consistent with the closed world assumption if the set of facts not derivable from the database is consistent with the database, taken as a theory. Definite databases (those not containing disjunctions) are consistent with the closed world assumption. In particular, databases containing disjunctions

This work was supported in part by contract N00039-82-G-0250 (the Knowledge Base Management Systems Project, Prof. Gio Wiederhold, Principal Investigator) from the Defense Advanced Research Projects Agency and by contract AFOSR-80-0212 (Universal Relations, Prof. Jeff Ullman, Principal Investigator) from the Air Force Office of Scientific Research, both of the United States Department of Defense, and by an AT&T Bell Laboratories Doctoral Scholarship. The views and conclusions contained in this document are those of the authors and should not be interpreted as representative of the official policies of DARPA or the US Government.

Authors' address: Computer Science Department, Stanford University, Stanford, CA 94305.

of multiple positive terms are not consistent with the closed world assumption. Under the closed world assumption, there is only one model of the theory, so there are no "maybe" statements. This is the usual model for database theories not containing nulls.

A third constraint, developed at length by Levesque [80, 82], may be called the *modified closed world assumption*. In this case, the theory may explicitly state where its knowledge is incomplete. The theory may contain disjunctions, and as in the open world assumption, it may have multiple models. All facts true in any particular model of a modified closed world theory must be derivable either as part of a disjunction explicitly mentioned in the theory, or else derivable from such disjunctions. All facts not derivable from such combinations of the disjunctions are assumed to be false. This assumption permits "true," "false," and "maybe" statements; however, many of the "maybe" statements in a given database under the open world assumption will become false under the modified closed world assumption. In a relational database, the disjunctions can appear at four levels: disjunctions of values, of tuples, of relations, and of databases. Definite database models of an indefinite database are obtained by choosing one of each of the disjuncts, provided that the resulting database satisfies all constraints.

In the work below, we restrict our attention to databases under the modified closed world assumption. Further, we will only consider the least expressive and most tractible levels of disjunction, those of values and of tuples with *true*.

Let us present some examples. Consider the following database.

Name	Address	Telephone
Susan	Apt 7 or 12	555-0123
Pat	Apt 7	555-9876
Sandy	Apt 17	none
George	Apt 9	unknown

Who is in Apt 7? The "true" result is Pat, and the "maybe" result is Susan.

Is Susan in Apt 7 or Apt 12? We would like to answer "yes"; after all, it is necessarily true that Susan may be found at one or both of these addresses. However, we have a potential problem in that this query is not equivalent to the disjunction of the queries "Is Susan in Apt 7?" and "Is Susan in Apt 12?"; for the answer to this disjunction is "maybe." The query answering algorithm must expend particular effort to deduce the "yes" answer rather than the "maybe" answer.

Who does not have a phone starting with 555? The "true" result is Sandy, and the "maybe" result is George.

2. Incompleteness and Relational Databases

How can the relational model [Codd 70, 79, 82, Maier 83, Ullman 83] be extended to include incomplete information? Let us assume that the relational model is capable of representing the relevant portion of the real world, were the necessary information available. Then we shall explore extensions to the relational model to support various levels of incompleteness.

The standard relational model consists of a set of relation schemas and a set of constraints. Each relation schema has a set of labelled domains called *attributes*. A relation is an unordered set of tuples, each tuple assuming a value for each attribute. We will use the term *attribute value* to refer to the value of a particular attribute for a specified tuple. First normal form requires that each attribute for each tuple be an atomic value, that is, one value in its domain.

A simple incompleteness that may exist is that we may have only partial information available about an entity whose identity we know. If entity names may serve as keys to the relation, this corresponds to a tuple with a known key value, but with non-atomic values for some attributes. This situation violates first normal form in that we cannot assign a specific value to every attribute in the corresponding tuple.

Let us consider the types of such non-atomic values. The ANSI/X3/SPARC study group for database management systems specifications generated a list of 14 different manifestations of null values [ANSI 75], for which we propose a taxonomy as follows. First, it may be that no domain value is applicable for an attribute; consider, for example, the value of the attribute Supervisor's-Name for the president of a company. We call this value *inapplicable*. The second case occurs when the value is known to be in a particular set of values, perhaps including *inapplicable*. This concept of a *set null* includes null values specifed as ranges (for example, $20 < \text{Age} < 30$). Using an example from Section 1b, {Apt 7, Apt 12} is a set null. In the case where an attribute is applicable for a tuple but no further information is known, the set null is the entire domain of the attribute.

Note that the choice of sets as a representation formalism need not be restricted to null values: Any singleton set other than the value *inapplicable* represents a non-null value.* We may regard all occurrences of single values as degenerate cases of set nulls.

*Even when the nulls merely signify "no information," there are problems in answering queries to databases with nulls [Keller 84].

Almost all types of nulls considered in the literature are (possibly restricted) cases of set nulls.

2a. Objects

The concept of objects permits constraints on the appearance of inapplicable null values in relational databases [Goldstein 81, Maier 80, 83, Sciore 80]. In brief, a relation can be divided into a set of relations, all with the same key or primary attributes, so that desirable information can be recorded solely by creating tuples without *inapplicable* [Codd 79, El-Masri 79, 80, Wiederhold 83]. If the logical database design corresponds in this manner to the objects identified, and we assume that no null values are allowed in the primary attributes for an entity, we will never need the null value *inapplicable*. The possibility of an attribute being inapplicable for a given tuple can be handled by attaching a condition to the tuple, as described in the next section. In the discussion of updates and refinement, we assume that inapplicable nulls have been eliminated in this fashion.

2b. On Representation of Incompleteness

In this section, we will summarize ways of representing alternative worlds. A set of alternative worlds, each describable by a relational database, may also be described by a single database with conditions attached to tuples. However, it is difficult to compute solutions to queries for a database expressed in this form. Therefore, we also present other representations that are more conducive to manipulation.

Conditional relation This is the most expressive form for describing incompleteness in extended relational databases. A conditional relation is the extension of an ordinary relation to contain one additional attribute, a condition to be applied to each tuple. A tuple with a condition appended is called a *conditional tuple*, and it may appear in query "maybe" results.

We can identify several classes of simple, useful conditions. The *possible* condition is for tuples where no specific information is known about the conditions under which the tuple will exist. In other words, the existence of a *possible* tuple is independent of the state of the remainder of the database.

A second class consists of sets of *alternative* tuples. A set of alternative tuples is called an *alternative set*. Precisely one of the members of an alternative set must exist in any model of an incomplete database. Alternative tuples are simply a generalization of null values to null tuples, of set nulls to set tuples. In contrast to *alternative* tuples, any number of *possible* tuples may hold in any alternative world.

Another class of conditions—called *predicated*—is explored by Imielinski and Lipski [81]. It consists of expressions built up from atomic conditions using conjunction, disjunction, and negation. The atomic forms are *true*, *false*, and comparisons between an attribute and a definite value or between two attributes. The largest class—called *arbitrary*—consists of any relational expression that can be applied to ordinary relational databases.

In this paper we will restrict our attention to *possible* conditions. Due to space limitations, we cannot also cover *alternative* conditions.

Set nulls The inclusion of set nulls in conditional relations makes it possible to represent the relations more concisely and makes it easier to compute answers to queries, without increasing the expressive power of conditional relations.

Predicates Predicates are *data-dependent* constraints applied to null values. For the purposes of this paper, the most useful predicate is *marked nulls*, which denote known equality of the actual, unknown values of nulls. Two marked nulls with the same marking are known to have the same actual, unknown value, but two marked nulls with differing marks may or may not have the same actual, unknown value. More complex predicates are also possible, but we shall not consider them here.

3. Updating Incomplete Databases That Model Static Worlds

Each incomplete database corresponds to a set of alternative worlds that the database models. When we update an incomplete database, the update will fall into one of two categories. The first type of update adds previously unknown information about a static world situation and generates a new set of alternative worlds that is a subset of the original group of alternative worlds. This type of update is relatively easy to implement correctly. The second type of update involves tracking changes in a dynamic world of which we have incomplete knowledge. Static world knowledge-adding updates are discussed in this section, and the discussion of dynamic world updates is deferred to Section 4.

It is important to note that under the modified closed world assumption, any entity that is known to possibly participate in a relation should be represented by a separate tuple in that relation, perhaps with a *possible* condition attached. Otherwise, the introduction of any tuple that implies the participation of that entity in the relation must be treated as a dynamic world change-recording update.

There are two concerns in updating an incomplete database that *models* a static world. The first of these, the nature of the updates and how to specify them, will be discussed in Section 3a. The second concern is how to assimilate the update into the database via *refinement*. Refinement has appeared in the literature [Imielinski 83, Maier 83, Osborn 81, Walker 80], and it will be considered in Section 3b.

3a. Updates in a Static World

Updates in incomplete databases modelling static worlds serve to *add knowledge* to the database. There may be many alternative worlds satisfying an incomplete static world database, and updates may reduce the number of these possibilities. For example, conditional tuples can have their conditions determined or augmented. Set nulls can be updated by eliminating some alternatives from the sets. Additional predicates, such as marked nulls, can be imposed on the database.

The first step in processing an update is to determine the "true" and "maybe" results of its selection clause. This is a difficult problem which will not be treated here (see [Codd 79, Maier 83]); the discussion below presupposes a successful resolution of the query answering problem. We note that syntactic extensions to the language, with accompanying semantic definitions, will be needed to designate set nulls. In addition, the user must be able to add and remove *possible* conditions in updates in order to satisfy the requirements of the modified closed world assumption and our postulations regarding the use of *inapplicable* null values.

Under the modified closed world assumption, deletions have no place in a static world. A tuple update consisting of a deletion followed by an insert operation will violate the modified closed world assumption unless the two are bundled into the same transaction. We use the convention that an **UPDATE** operation specifies the modification of an entity or relationship already in the database, while an **INSERT** operation supplies information about a new entity or relationship. In a static world under the modified closed world assumption, **UPDATE** requests are only reasonable to the extent that they supply additional, non-conflicting information about existing entities; **INSERT** requests are not permitted, for there can be no new entities.

With the **UPDATE** operator, one may update the "true" results of a selection clause as usual, with some extra attention given to handling marks. But what action should be taken on the "maybe" result of the selection clause? In a static world under the modified closed world assumption, an update can only serve to narrow the range of choices within a set null. Therefore, the first possibility is that the target attribute values do not already include the new values, in which case the tuple cannot be in the "true" result of the selection clause. A sophisticated query processor might use that fact to refine certain fields of the failing tuple. The second possibility is that the target attribute values do already include the new values, in which case the best action in our model is simply to ignore the update. The third possibility is that the old and new attribute values are set nulls with a partial overlap in values; in this case we may try a simple technique called *tuple splitting*. Consider the following example.

Vessel	HomePort	*Condition*
{Henry, Dahomey}	{Boston, Charleston}	*true*

```
UPDATE [HomePort := SETNULL ({Boston, Cairo})]
WHERE Vessel = "Henry"
```

In the result, we have split the original tuple into two *possible* tuples. One tuple covers the case that the tuple is actually in the "true" result, and the other tuple covers the "false" result case. If we perform the update, we could get the following relation.

Vessel	HomePort	*Condition*
{Henry, Dahomey}	{Boston, Cairo}	*possible*
{Henry, Dahomey}	{Boston, Charleston}	*possible*

Note, however, that the Henry could *not* be in Cairo because that was not permitted in the original database, and we are working in a static world under the modified closed world assumption. This gives us the following result.

Vessel	HomePort	*Condition*
{Henry, Dahomey}	Boston	*possible*
{Henry, Dahomey}	{Boston, Charleston}	*possible*

It is a very difficult problem in general to determine exactly which set null values would put a tuple in the "true" result and which would put it in the "false" result. However, a smarter query answering algorithm might be able to produce the following.

Vessel	HomePort	*Condition*
Henry	Boston	*possible*
Dahomey	{Boston, Charleston}	*possible*

Since there may now be zero, one, or two ships, this method violates the modified closed world assumption in a static world. This problem may be avoided by using an alternative set containing the two tuples, so that precisely one of them will hold. This latter approach incurs other problems that are beyond the scope of this paper.

3b. Refinement in a Static World

Refinement is a process that alters the state of the database without affecting its set of possible worlds. If coupled with a query answering strategy that generates all possible worlds and then performs the query on each of them, refinement may affect the efficiency of the computation but not the answers to queries; otherwise, refinement may affect the answers.

Refinement simplifies the contents of the database by applying known dependencies and constraints to establish conditions on the existence of null values [Vassiliou 80, Lien 79, Maier 83]. This process may allow a query answering strategy to provide more informative answers to queries. Refinement can also help to verify the compatibility of updates with static world databases, thus preventing violations of dependencies and integrity constraints.

Let us begin by considering functional dependencies. In our model, we can use these dependencies to establish when two nulls must have the same mark. For example, suppose Ship → HomePort in the following relation.

Ship	HomePort
Wright	{Managua, Taipei}
Wright	{Taipei, Pearl Harbor}

We may conclude that this is actually the following relation.

Ship	HomePort
Wright	Taipei

We have eliminated a null value, enabling us to give more informative answers to queries. For example, if the user asks for a list of all ships with a HomePort of Taipei, then the Wright will be in the "maybe" result for the unrefined database, but in the "true" result for the refined version. More generally, suppose we are given a relation with $A \rightarrow B$, containing two tuples with set nulls s_1 and s_2, as follows.

A	B
a1	s_1
a1	s_2

We may refine this to the following single tuple.

A	B
a1	$s_1 \cap s_2$

Similarly, if $A \rightarrow B$ in the following relation, and b1 and b2 are known to be unequal, then we may conclude that a1 and a2 must have different values. Indeed, either or both of b1 and b2 may be set nulls, as long as the sets have no elements in common.

A	B
a1	b1
a2	b2

If, say, a1 is a non-null value, then we can replace a2 by a2 − a1. That is, the keys of the two tuples must be unequal.

Functional dependencies can also be used to refine the conditions appended to tuples. For example, let $A \rightarrow B$ in the following relation.

A	B	*Condition*
a1	b1	*true*
a1	b1	*possible*

If a1 is a non-null value, this refines to the following relation.

A	B	*Condition*
a1	b1	*true*

Refinement helps to catch consistency errors that are violations of known dependencies. (The refinement process is similar to the chase algorithm for inference of dependencies [Ullman 83].) The presence of such errors is signalled by the appearance of a set null with no elements (the empty set). For example, if $s_1 \cap s_2$ in the example above is the empty set, then an error has occurred. As presented, refinement is not sufficient to detect all violations of functional dependencies, nor to eliminate as many nulls as would be possible with a more general mechanism.

We have given some simple rules for refining databases with functional dependencies. One may define rules in a similar fashion for all varieties of generalized dependencies.

4. Tracking Changing Worlds

Let us now consider the situation where the database is modeling a dynamic, changing world. We will discuss issues relating to updates and refinement in this context.

4a. Updates in a Changing World

In the discussion below, we consider how to insert tuples to relations, how to delete them, and how to do other kinds of updates. These updates will fall into two categories: *knowledge-adding updates*, which represent new information about the dynamic world at one particular moment in time, and *change-recording updates*, which track changes in the world over time. We will consider corrections as knowledge-adding updates if the new set of possible worlds is included in the original; otherwise they are change-recording updates because they cause a transformation to a different set of possible worlds. Equivalently, before performing a knowledge-adding update, the database already models the new set

of possible worlds. Change-recording updates are particularly difficult to execute correctly, and matters are complicated by the fact that it is not usually possible to tell whether an update is knowledge-adding or change-recording. Knowledge-adding updates were discussed in the previous section; here we consider change-recording updates and how to proceed when the type of an update is unknown.

For example, consider the following relation and update.

Vessel	Port	Cargo
Dahomey	Boston	Honey
Wright	{Boston, Newport}	Butter

```
INSERT [Vessel := "Henry", Cargo := "Eggs",
   Port := SETNULL ({Cairo, Singapore})]
```

The result after performing the update on the relation is as follows.

Vessel	Port	Cargo
Dahomey	Boston	Honey
Wright	{Boston, Newport}	Butter
Henry	{Cairo, Singapore}	Eggs

Under the modified closed world assumption, this is a change-recording update because the Henry was not previously known to exist. Unfortunately, change-recording insert operations can interact disastrously with refinement in relations with functional dependencies. (We will discuss this further in Section 4b.)

We use the convention that an UPDATE operation specifies the modification of an entity or relationship already in the database, while an INSERT operation supplies information about a new entity or relationship. It is not always clear whether an assertion should be treated as an insertion or an update. This is especially true when the update is specified through natural language [Davidson 84].

For other types of updates, tuples in the "true" result of the selection clause can be updated as usual. For "maybe" results, when only set nulls are involved, the first option is to do nothing and expect the user to explicitly update the "maybe" result by means of a truth operator in the selection clause [Codd 79, Lipski 79], as in the following example.

```
UPDATE [Port := Cairo]
WHERE MAYBE (Port = "Cairo")
```

Result:

Vessel	Port	Cargo
Dahomey	Boston	Honey
Wright	{Boston, Newport}	Butter
Henry	Cairo	Eggs

As a second option, the database system can explicitly ask the user on the fly what to do about the "maybe" results.

As a third option, we can bravely attempt to automatically update the "maybe" results. In a model of conditional tuples and set nulls, one can use the tuple-splitting technique of Section 3b. Consider the effect of a cargo update on the previous relation.

```
UPDATE [Cargo := "Guns"]
WHERE Port = "Boston"
```

Vessel	Port	Cargo	*Condition*
Dahomey	Boston	Guns	*true*
Wright	{Boston, Newport}	Guns	*possible*
Wright	{Boston, Newport}	Butter	*possible*
Henry	Cairo	Eggs	*true*

We have given the original tuple a *possible* condition, created a duplicate, and then performed the update in place on the new tuple. (The two null values {Boston, Newport} would be given the same mark.) We have generated quite a few new alternative worlds for the database. To reduce this diversification, a clever query answering algorithm might be able to tell us which set null values would give rise to "false" result tuples and which to "true" result tuples. With such an algorithm, we could give the following result from the cargo update.

Vessel	Port	Cargo	*Condition*
Dahomey	Boston	Guns	*true*
Wright	Boston	Guns	*possible*
Wright	Newport	Butter	*possible*
Henry	Cairo	Eggs	*true*

As we have seen, appending *possible* conditions when splitting tuples generates new possible worlds. The use of an *alternative* set for the split tuples avoids this problem at the expense of additional complications during future updates, a consideration beyond the scope of this paper.

Another potential solution is *null propagation*, where fields that are the target of an update are transformed into set nulls. The following series of examples illustrates this technique.

AB	*C*
A B	C
v1 {v2, v3}	v2
	v3

```
UPDATE [A := C]
WHERE B = C
```

Using null propagation, we obtain the following relation

AB.

A	B
{v1, v2}	{v2, v3}
{v1, v3}	{v2, v3}

However, the set of possible worlds corresponding to this database is disjoint from the correct set of possible worlds. Splitting the original tuple into two *alternative* tuples, we obtain the following relation *AB* (before the update).

A	B	*Condition*
v1	v2	*alternative set 1*
v1	v3	*alternative set 1*

The updated relation then becomes the following.

A	B	*Condition*
v2	v2	*alternative set 1*
v3	v3	*alternative set 1*

To delete a tuple that is in the "maybe" result, one could append the *possible* condition and refine the tuple. Consider the following relation and update.

Ship	Port
{Jenny, Wright}	{Boston, Cairo}

```
DELETE WHERE Ship = "Jenny"
```

With a somewhat clever query algorithm, we can first split the tuple as follows.

Ship	Port	*Condition*
Jenny	{Boston, Cairo}	*alternative set 1*
Wright	{Boston, Cairo}	*alternative set 1*

We then delete the first tuple as requested. Notice that the second tuple changes from an *alternative* tuple to a *possible* tuple.

Ship	Port	*Condition*
Wright	{Boston, Cairo}	*possible*

Deletion under the modified closed world assumption is a very strong statement; a deletion based on a key value is equivalent to declaring that the entity is no longer in the world. To delete a relationship between entities that continue to exist, it is better to replace the original relationship with one or more relationships containing nulls. If this is done, the original entities will continue to be known, but they will be unrelated.

In the next section, we will consider refinement in the face of change-recording updates.

4b. Refinement in a Changing World

In a static world, refinement is a safe process; in a dynamic world, refinement must only be done at a correct static state. For a correct final dynamic world database state to be achieved, conflicting updates must be supplied in the right order, and refinement must not be done until all change-recording updates corresponding to the same point in time have been accepted—in other words, until the database corresponds to an actual static world state.

In a static world, refinement does not affect the set of possible worlds; rather, it affects the difficulty of computing the set of possible worlds. On the other hand, updates can reduce the set of possible worlds. In a static world, a refined database is *equivalent* to its unrefined version, in that they give the same answers to all queries. Note, however, that refinement may assist a query answering strategy to produce a more informative answer, even though the databases are equivalent.

In a dynamic world, refinement may affect the set of worlds possible after an update. Given a database, refinement can produce a second equivalent database, but after identical updates, the refined and unrefined updated databases may no longer be equivalent [Fagin 83, Kuper 84].

The problem with refinement can arise when attempting to store facts that are general rules in the database along with facts that are merely statements of current conditions. For example, suppose that the Kranj and the Totor alternate between Victoria and Vancouver. Thus, one of these two ships is always in Vancouver. However, we also know what the Totor is currently in Victoria. This results in the following relation.

Ship	Location
{Kranj, Totor}	Vancouver
Totor	Victoria

The relation after refinement follows.

Ship	Location
Kranj	Vancouver
Totor	Victoria

Suppose that the Totor moves to Vancouver. This changes the relation as follows.

Ship	Location
Kranj	Vancouver
Totor	Vancouver

But if we apply the update to the unrefined relation, we get a different relation.

Ship	Location
{Kranj, Totor}	Vancouver
Totor	Vancouver

Notice that this relation admits the possibility that the Kranj has moved to Victoria. This example shows that

refinement can cause some worlds not to be possible when the database undergoes a change-recording update.

5. Summary and Conclusion

We have considered how databases can be used to model incomplete knowledge about the world. Given a body of incomplete knowledge, there is a set of possible worlds that are consistent with that knowledge. The modified closed world assumption allows the database to explicitly state where its information is incomplete. If the database provides a definite answer to a query, we want that answer to hold in the real world that the database is modeling. Where knowledge is lacking, the database may have to indicate that the answer is "maybe." We would like the database to provide definite answers whenever possible. An answer to a query which is true in all possible world models of the database is considered the "true" result. Similarly, the "false" result is not true in any possible world. That which is true in some worlds and false in others is in the "maybe" result. Some query answering strategies may not be able to find all the "true" and "false" results to some queries, and instead report an expanded "maybe" result.

We have considered extensions to the relational database model to support incompleteness. Conditional relations, which consist of tuples whose existence is dependent on some condition, are sufficient to express all incompleteness that can be modeled by a set of alternative worlds each completely expressible by a relational database. Unfortunately, generating alternative worlds or answering queries for conditional relations is quite complex. On the other hand, set nulls present a method for handling incomplete information for which simpler query answering strategies exist. However, set nulls alone do not have the expressive power of conditional relations. The expressive power of set nulls can be enhanced by using predicates which the database must satisfy. Equality predicates, usually modeled by marked nulls, are one important form. Another useful extension is to allow sets of alternative tuples. Exactly one tuple of each such set must exist in any model of the database.

In updating databases that model incomplete worlds, we distinguish between knowledge-adding updates, which reduce the set of possible worlds, and change-recording updates, which reflect other changes in the set of possible worlds. The former may be described as providing a more complete model of a static world; they merely narrow down the set of possible worlds. On the other hand, a change-recording update marks a transition to a new set of possible worlds, and is very difficult to perform with an acceptable degree of precision. This problem is exacerbated by the interactions between refinement and change-recording updates.

6. Acknowledgements

Jeff Ullman and Gio Wiederhold provided advice, encouragement, and support. Christos Papadimitriou gave extensive constructive criticism and suggestions.

7. Bibliography

[ANSI 75] "ANSI/X3/SPARC Study Group on DBMSs Interim Report," in *SIGMOD FDT Bulletin*, 7:2, 1975. (Fourteen reasons for null values also in Atzeni and Parker, "Assumptions in Relational Database Theory," in *Proc. of the ACM Symp. on Principles of Database Systems*, (Los Angeles), March 1982.)

[Chamberlin 75] D. D. Chamberlin, J. N. Gray, I. L. Traiger, "Views, Authorization, and Locking, in a Relational Data Base System," *Proc. National Computer Conference*, AFIPS, 1975.

[Codd 70] E. F. Codd, "A Relational Model for Large Shared Data Banks," *Comm. of the ACM*, 13:6, June 1970, pp. 377-387.

[Codd 79] E. F. Codd, "Extending the Database Relational Model to Capture More Meaning," *ACM Trans. on Database Systems*, 4:4, December 1979.

[Codd 82] E. F. Codd, "Relational Database: A Practical Foundation for Productivity," *Comm. ACM*, 25:2, February 1982. This is Codd's 1981 Turing Award Lecture.

[Davidson 84] James E. Davidson, "A Natural Language Interface for Performing Database Updates," *IEEE Computer Soc. Computer Data Engineering Conf.*, (Los Angeles), April 1984.

[Dayal 82] U. Dayal and P. Bernstein, "On the Correct Translation of Update Operations on Relational Views," *ACM Trans. on Database Systems*, 7:3, September 1982.

[El-Masri 79] Ramez El-Masri and Gio Wiederhold, "Data Models Integration using the Structural Model," *Proc. of the 1979 SIGMOD Conference*, ACM SIGMOD, Boston, June 1979.

[El-Masri 80] Ramez El-Masri, *On the Design, Use, and Integration of Data Models*, Ph.D. dissertation, Stanford University, 1980.

[Fagin 83] Ronald Fagin, Jeffrey D. Ullman, and Moshe Y. Vardi, "On the Semantics of Updates in Databases," *Proc. of the Second ACM Symp. on Principles of Database Systems*, (Atlanta, GA), March 1983.

[Goldstein 81] Billie S. Goldstein, "Constraints on Null Values in Relational Databases," in *Proc. 7th Int. Conf. on Very Large Data Bases*, (Cannes, France), September 1981.

[Imielinski 81] Tomasz Imielinski and Witold Lipski, Jr., "On Representing Incomplete Information in a Relational Database," in *Proc. 7th Int. Conf. on Very Large Data Bases*, (Cannes, France), September 1981.

[Imielinski 83] T. Imielinski and W. Lipski, Jr., "Incomplete Information and Dependencies in Relational Databases," in *Proc. of Annual Meeting: SIGMOD and Database Week*, (San Jose, CA), May 1983; the proceedings appeared as *SIGMOD Record*, 13:4, ACM, May 1983.

[Keller 82] Arthur M. Keller, "Updates to Relational Databases Through Views Involving Joins," in *Improving Database Usability and Responsiveness*, Peter Scheuermann, ed., Academic Press, New York, 1982.

[Keller 84] Arthur M. Keller, "Some Problems of Null Completion in Relational Databases," submitted for publication.

[Kuper 84] Gabriel M. Kuper, Jeffrey D. Ullman, and Moshe Y. Vardi, "On the Equivalence of Logical Databases," *Proc. of the Third ACM Symp. on Principles of Database Systems*, (Waterloo, Ontario, Canada), April 1984.

[Levesque 80] H. J. Levesque, "Incompleteness in Knowledge Bases," *Proc. of the Workshop on Data Abstraction, Databases and Conceptual Modeling*, (Pingree Park, Co.), June 1980, also *ACM SIGART Newsletter*, No. 74, January 1981.

[Levesque 82] H. J. Levesque, "A Formal Treatment of Incomplete Knowledge Bases," Computer Sys. Research Group, Tech. Rep. CSRG-139, Univ. of Toronto, Ph.D. diss., February 1982.

[Lien 79] Y. Edmund Lien, "Multivalued Dependencies with Null Values in Relational Data Bases," in *Proc. 5th Int. Conf. on Very Large Data Bases*, (Rio de Janiero, Brasil), October 1979.

[Lipski 79] W. Lipski, Jr., "On Semantic Issues Connected with Incomplete Information Databases," *ACM Trans. on Database Systems*, 4:3, September 1979.

[Maier 80] D. Maier, "Discarding the Universal Instance Assumption: Preliminary Results," *Proc. XP1 Workshop on Relational Database Theory*, (Stony Brook, NY), July 1980.

[Maier 83] D. Maier, *Theory of Relational Databases*, Computer Science Press, Rockville, MD, 1983. (Especially Chapter 12, "Null Values, Partial Information, and Database Semantics.")

[Osborn 81] S. Osborn, "Insertions in a Multi-relation Database With Nulls," *Proc. of COMPSAC 81: IEEE Computer Society's Fifth Int. Computer Software and Applications Conference*, (Chicago), November 1981.

[Reiter 78] Raymond Reiter, "On Closed World Databases," in *Logic and Databases*, Gallaire and Minker, eds., Plenum, New York, 1978.

[Reiter 80] Raymond Reiter, "Data Bases: A Logical Perspective," in *Proc. Workshop on Data Abstraction Databases and Conceptual Modeling*, (Pingree Park, CO), June 1980, appeared as *SIGMOD Record*, 11:2, February 1981.

[Sciore 80] Edward Sciore, *The Universal Instance Assumption and Database Design*, Ph.D. dissertation, Princeton University, October 1980.

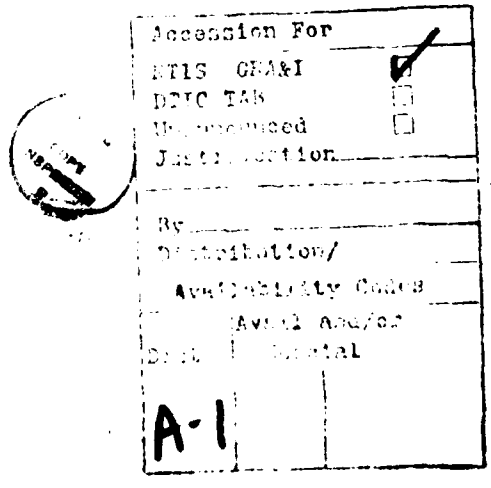
[Stonebraker 75] Michael Stonebraker, "Implementation of Integrity Constraints and Views by Query Modification," *Proc. of the 1975 SIGMOD Conference*, (San Jose), June 1975.

[Ullman 83] Jeffrey D. Ullman, *Principles of Database Systems*, Computer Science Press, Potomac, MD, second edition, 1983.

[Vassiliou 80] Yannis Vassiliou, "Functional Dependencies and Incomplete Information," in *Proc. 6th Int. Conf. on Very Large Data Bases*, (Montreal), October 1980.

[Walker 80] A. Walker, "Time and Space in a Lattice of Universal Relations with Blank Entries," *Proc. XP1 Workshop on Relational Database Theory*, (Stony Brook, NY), July 1980.

[Wiederhold 83] Gio Wiederhold, *Database Design*, McGraw-Hill, New York, second edition, 1983.




ATE
LME