DYNAMIC CONSTRAINTS AND DATABASE EVOLUTION

(Extended Abstract)

by

Victor Vianu*

University of Southern California

Department of Computer Science

Los Angeles, CA 90089-0782

Abstract.

A simple extension of the relational model is introduced to study the effects of dynamic constraints on database evolution. Both static and dynamic constraints are used in conjunction with this "dynamic" extension of the relational model. The static constraints considered here are functional dependencies (fd's). The dynamic constraints involve global updates and are restricted to certain analogs of fd's, called "dynamic" fd's. The results concern the interaction between the static and dynamic constraints. The effect of the past history of the database on the static constraints is investigated using the notions of age and age-closure. The connection between the static constraints and the future evolution of the database is described through the notions of survivability and survivability-closure.

1. Introduction.

One important aspect of understanding objects in the real world concerns the laws that govern their evolution in time. Many database systems capture this essential semantic component by imposing "dynamic" constraints on how the database can be changed [Bro, CePBra, DeAz, HaM]. Previous work on this topic has focused mainly on expressing wide classes of dynamic constraints using a logic-based formalism [CaFu, CoCaFu, ClW, G, NYaz]. Missing from this predominantly descriptive approach has been an investigation of the effects of dynamic constraints on database evolution. This paper is a first contribution to such an effort. The purpose of this paper is to introduce a simple, formal model for the evolution of databases in time, and use this model to study the effects on database evolution of a significant (but tractable) type of dynamic constraint.

Our model is a simple extension of the relational model. In this "dynamic relational model", a database is viewed as a sequence of instances in time. Each new instance is obtained from the previous one by updates, insertions, and deletions. The fact that tuples preserve their identity through updates is formally expressed. Two types of constraints are used in conjunction with database sequences static and dynamic. The static constraints (imposed on each instance in a sequence) are restricted here to functional dependencies (fd's). The dynamic constraints (which relate each instance in the sequence to its successor) are focused on global updates, i.e., updates affecting several (or all) tuples in the

*The author was supported in part by the National Science Foundation under grant number MCS-7925004.

database simultaneously. These constraints are restricted to certain "dynamic" analogs of fd's.

Our results focus primarily on inferring static constraints from knowledge about the evolution of the database, as expressed by the dynamic constraints. After giving some motivating examples (§2) we formally present the model and define the static and dynamic constraints used in conjunction with it (§3). In §4 we show how to infer static constraints in the context of a single update satisfying given dynamic constraints. Then we define the notion of "age" (which summarizes the past history of the database) and describe its effect on static constraints (§5). Finally we define "survivability" and "potential immortality" (which reflect the potential future evolution of the database) and describe their effect on the static constraints (§6). In this abstract, some definitions and results are stated informally and others are omitted.

The dynamic relational model and the particular types of constraints studied here are motivated by their simplicity and frequent occurrence in real situations. Both can be extended in several directions. For example, more general types of constraints, such as Fagin's "embedded implicational dependencies" [F], can be used instead of fd's as the basis for dynamic constraints. The dynamic constraints on global updates can be augmented with constraints on individual tuple updates, insertions, and deletions. And we may consider dynamic constraints which involve non-relational database models (e.g. the "Format Model" of [HuYap]), or are model independent (in the spirit of Spyratos [S]). Questions analogous to those raised in this paper can be asked about any of the above extensions, and new questions are likely to arise.

This paper is based on portions of [V]. Further details and discussion, as well as the proofs, can be found there. Additional topics are investigated in [V] with respect to the model, such as user views obtained by the operation of projection, and view updates.

2. Motivating Example.

We present here two examples which illustrate our dynamic constraints and related ideas. For the time being we use informal notation and terminology. The formal definitions will be presented in §3.

2.1 Example. Consider an "employee" database containing a relation EMP with attributes

ID#, NAME, SEX, AGE, RACE, MERIT, SAL.

The ID# uniquely identifies each employee, MERIT is an integer between 0 and 10 which reflects the performance of the employee, 10 being best[1], SAL is the salary. Since ID# is a key, the static constraint

(s1) ID# -> NAME SEX AGE RACE MERIT SAL

must always hold. Suppose that it is company policy to have an annual salary increase on each July 1st. At that time the database is globally updated to reflect changes in salary. When this occurs, the new values of certain attributes are related to the old values of other attributes. For example, we expect the old ID# of an employee to determine the new ID#.[2] For each attribute A, let $\check{A}$ denote its old value and $\hat{A}$ its new value. Then the above can be expressed as a "dynamic" functional dependency by $\check{ID}\# \rightarrow \hat{ID}\#$. In addition, no two employees (with different ID#'s) should receive the same new ID# as a result of an update. Thus, the dynamic fd $\hat{ID}\# \rightarrow \check{ID}\#$ must also hold. (We will abbreviate two such dynamic fd's by $\check{ID}\# <-> \hat{ID}\#$.) Similar arguments can be made for NAME, SEX, RACE, and AGE. Thus, the following dynamic fd's hold at each update

(d1) $\check{ID}\# <-> \hat{ID}\#$, $\check{NAME} <-> \hat{NAME}$, $\check{SEX} <-> \hat{SEX}$, $\check{RACE} <-> \hat{RACE}$, $\check{AGE} <-> \hat{AGE}$.

Let us look at two possible policies relating to salary increases. The first is the "equal opportunity" policy, where each new salary is

---

[1] We assume that MERIT can be freely updated at any time.

[2] It is conceivable that the value of ID# may occasionally change. For example, U.S.C. uses temporary identification numbers for its foreign employees who do not have a social security number yet.

determined solely by merit in conjunction with the old salary. This is reflected in the database by the satisfaction of the additional dynamic fd

(d2) MERIT SAL -> SAL.

The second policy is the "discriminatory" policy, where each salary is determined strictly on the basis of sex, race, and age. Then each update satisfies, instead of (d2), the dynamic fd

(d3) SEX RACE AGE -> SAL.

Now let us look at the interaction between the static constraints (s1) and the various sets of dynamic constraints (d1), (d1,d2), and (d1,d3). Suppose EMP satisfies (s1) and is updated according to (d1). Due to the dynamic fd's ID <-> ID, it is easily seen that (s1) is satisfied automatically by the new EMP. Also, it can be seen that no additional static constraints must hold. If EMP is updated according to (d1,d2) - the equal opportunity policy - the same situation occurs. Now suppose EMP is updated according to (d1,d3) - the discriminatory policy. As earlier, (s1) is preserved in the new EMP. In addition however, the new version of EMP must satisfy the static fd

(s2) SEX RACE AGE -> SAL.

[This is so because SEX RACE AGE -> SEX RACE AGE and SEX RACE AGE -> SAL hold. Then SEX RACE AGE -> SAL holds by transitivity. A formal way to compute the static fd's implied in an update is provided by "dynamic mappings", which are introduced in Section 4.] In fact, each instance of the database must satisfy (s2) after at least one update has occurred.

Philosophically, there is an important difference in the nature of the dynamic constraints reflected in (d1) and those reflected in (d2) and (d3). The first group, which we informally refer to as "intrinsic", arises from the very nature of the object considered (employee) and is inseparable from our understanding of that object. (For example, an employee cannot change his race.) This is not the case with the second group, called "extrinsic". Indeed, (d2) and (d3) are the result of certain outside variable factors, in this case human policy. Intuitively, intrinsic dynamic constraints should not produce any additional static constraints, since (assuming we have a complete understanding of the object) any such "consequence" should have been anticipated in the first place. However, certain static constraints could be preserved by updates. An example is provided by (d1) which preserves (s1) but does not imply any new static fd's. On the other hand, "extrinsic" dynamic constraints can be expected to imply additional static constraints which arise from the particular type of evolution imposed on the object.[3] For example, this is the case with (d3) which implies the new static fd (s2).

In the simple example above, the static consequences are easy to anticipate. In particular, they become apparent after a single update. This is not always the case, as will be seen in the next example. (In fact, it is shown in Section 5 that it can take arbitrarily long for all static consequences of the evolution to become present in the database.)

2.2 Example. Consider the following database, used by the Credit and Loans Division of a bank. (For simplicity, many real-life details are omitted.) The database contains a relation called CAMPAIGN, which is used as part of an annual campaign to decide what loans and credit card offers are mailed out to each of the bank's customers. This is done according to a fixed policy, detailed below. The (relevant) attributes are

ACCT#: Account number.

BAL: The value of BAL is 0 if the customer's savings account balance is under $2000 and 1 if it is over $2000.

MC, VISA, AMEX Each domain is equal to {YES, NO}. The value is YES if the customer has been offered a Master Charge Card (MC), VISA card (VISA), or American Express card (AMEX). The value is also YES when a tuple representing a new customer is first inserted in the database, if the customer has the card from his old bank. In all other cases, the value is NO.

---

[3] In other words, specific evolutionary conditions may generate objects endowed with specific additional qualities. Note that similar observations are made in [Da].

CAR, RV, HOUSE Each domain is equal to {YES, NO}. The value is YES iff the customer has been offered a car loan (CAR), an RV loan (RV) or a house loan (HOUSE).

The database is updated annually, each time a mail campaign occurs.

Let us consider the static and dynamic fd's satisfied by the CAMPAIGN relation. First note that the static fd

(s1) ACCT# -> BAL MC VISA AMEX CAR RV HOUSE

always holds, since ACCT# is a key. For the same reason, the dynamic fd's

(d1) $\check{\text{ACCT\#}} <-> \hat{\text{ACCT\#}}$

always hold in any update.

Now let us look at the bank's policy with regard to the credit cards and loans offered to customers, and the dynamic fd's which are a consequence of this policy.

1. The credit division has the following rules concerning credit cards [4]

   (a) A Master Charge card is offered to any customer with a savings account balance of at least $2000. This is reflected by the dynamic fd's

   $\check{\text{BAL}} <-> \hat{\text{MC}}$.

   (b) A VISA card is offered to each person who was offered a Master-Charge card last year (or became a customer then, and already had an MC) and is still a customer of the bank. Thus, the dynamic fd's

   $\check{\text{MC}} <-> \hat{\text{VISA}}$

   hold.

   (c) An American Express Card is offered to each person who was offered a VISA card last year (or became a customer then, and already had a VISA) and is still a customer of the bank. Hence the dynamic fd's

   $\check{\text{VISA}} <-> \hat{\text{AMEX}}$

   hold.

2. The loans division has the following rules

   (a) A car loan is offered to all customers with a balance of at least $2000. This is reflected by the dynamic fd's

   $\check{\text{BAL}} <-> \hat{\text{CAR}}$.

   (b) An RV loan is offered to all persons who were offered a car loan last year and are still customers of the bank. Thus, the dynamic fd's

   $\check{\text{CAR}} <-> \hat{\text{RV}}$

   hold.

   (c) A house loan is offered to all persons who were offered RV loans last year and are still with the bank hold. Hence the dynamic fd's

   $\check{\text{RV}} <-> \hat{\text{HOUSE}}$.

   Altogether, the following dynamic fd's hold, in addition to (d1):

   (d2) $\check{\text{BAL}}<->\hat{\text{MC}}$, $\check{\text{MC}}<->\hat{\text{VISA}}$, $\check{\text{VISA}}<->\hat{\text{AMEX}}$, $\check{\text{BAL}}<->\hat{\text{CAR}}$, $\check{\text{CAR}}<->\hat{\text{RV}}$, $\check{\text{RV}}<->\hat{\text{HOUSE}}$.

   Let us look at the effect of the set of dynamic fd's (d1,d2) on the static constraints. In this example, the full static consequences of the dynamic constraints become apparent in the course of three updates. Indeed, it can be verified that·

3. If CAMPAIGN satisfies (s1), and is updated once according to (d1,d2) then (s1) is preserved in the new CAMPAIGN and the additional fd's

   (s2) MC <-> CAR

   hold.

4. If CAMPAIGN satisfies (s1) and is updated twice according to (d1,d2), then (s1) is preserved and the additional fd's

   (s3) MC <-> CAR and
   VISA <-> RV

   hold.

5. If CAMPAIGN satisfies (s1) and is updated three or more times according to (d1,d2), then (s1) is preserved and the additional fd's

   (s4) MC <-> CAR,
   VISA <-> RV, and
   AMEX <-> HOUSE

   hold. Thus the above fd's always hold for customers who were with the bank for at least 3 years after the credit and loan policies went into effect. [The tuples representing these customers are said to have attained age 3 in the database, i.e., were updated at least 3 times.]

[4] Since the cost of making offers is very low, the bank is not concerned about superfluous offers (e.g. to persons who already have the credit card). The bank's primary concern (and that reflected in the outlined policy) is to make offers only to customers who meet its eligibility requirements.

3. The Model.

In this section we present our dynamic relational model for the evolution of databases in time. Then we define the particular types of dynamic constraints that are studied here within the model. For the purpose of this abstract we assume the reader is familiar with the notions of attribute, relation, functional dependency[5] (fd) and satisfaction of fd's [Ul]. If U is a set of attributes, FD(U) is the set of all fd's over U. All attributes have infinite domains.

As mentioned in the introduction to the paper, our model is a simple extension of the relational model. Each database is viewed as a sequence of instances in time. Since all constraints used in the paper are intra-relational, it is assumed that each instance of the database consists of a single relation (however, the universal relation property is not assumed, and the results developed here can fruitfully be extended to multi-relational databases). Informally, a change of state is realized through.

1. Updates: These consist of changing some values of tuples already in the relation (however, tuples can be left unaltered in an update).
2. Insertions. These consist of inserting new tuples in the relation.
3. Deletions: These consist of removing tuples.

The above is formalized through the notion of a "database sequence", defined next.

Definition. A database sequence (dbs) over U is a sequence $\{(I_i, \mu_i)\}_{0 \leq i \leq n}$, where n is a non-negative integer, each I_i $(0 \leq i \leq n)$ is an instance over U, each μ_i $(0 \leq i < n)$ is a partial one-to-one mapping from I_i into I_{i+1}, and $\mu_n = \emptyset$.

Intuitively, a database sequence is a sequence of successive instances of the database, together with "update mappings" from each "old" instance to each "new" instance. Semantically, we assume that each tuple is a representation of an object. Since a tuple and its update represent the same object in time, each tuple preserves its identity in an update. This is formally captured by the update mappings. Thus, for each tuple x in I_i, $\mu_i(x)$ represents its updated version in I_{i+1} (in particular, $\mu_i(x) = x$ if x is unchanged). Also, each tuple of I_i not in dom μ_i is deleted and each tuple in I_{i+1} not in range μ_i is an inserted tuple.[6]

These remarks apply for $i < n$.

In this first investigation we focus on database sequences described by certain simple types of static and dynamic constraints. The static constraints are restricted to functional dependencies. As in Examples 2.1 and 2.2, the dynamic constraints focus solely on global updates and are similar in nature to functional dependencies. We define the dynamic constraints using the notions of an update and its associated "action relation."

Definition. Let U be a set of attributes. An update over U is a triple (I, μ, I'), where I and I' are relations over U and μ is a one-to-one (total) mapping from I onto I'.

If $\{(I_i, \mu_i)\}_{0 \leq i \leq n}$ is a dbs over U, then (dom μ_i, μ_i, range μ_i) is an update over U for each i, $0 \leq i < n$.

Our notion of "action relation" associated with an update is closely related[7] to that introduced in [NYaz]. Intuitively, the action relation is formed by concatenating each tuple in the "old" instance with its updated version. Before giving the formal definition we introduce some symbolism to denote the corresponding "versions" of the attributes, one for the "old" values and another for the "new" values.

Notation. For each attribute A, let $\check{A}$ and $\hat{A}$ be new attributes whose domain is the domain of A. For each set U of attributes, let $\check{U} = \{\check{A} | A \in U\}$ and $\hat{U} = \{\hat{A} | A \in U\}$. For each tuple x over U let $\check{x}$ and $\hat{x}$ be

---

[5] We consider only fd's with nonempty left-hand sides.

[6] As customary, dom f and range f denote the domain and range of the mapping f, respectively.

[7] In [NYaz], as in this paper, the marked attributes of action relations are used to denote old and new values of attributes. In [NYaz] the emphasis is on showing how action relations can be used to express a large class of constraints on updates. Here we focus on a tractable constraint and explore its properties.

the tuples over $\check{U}$ and $\hat{U}$ respectively such that $\check{x}(\check{A}) = \hat{x}(\hat{A}) = x(A)$ for each $A \in U$. Finally, for each $\Sigma \subseteq FD(U)$ let $\check{\Sigma} = \{\check{X} \to \check{Y} | X \to Y \in \Sigma\}$ and $\hat{\Sigma} = \{\hat{X} \to \hat{Y} | X \to Y \in \Sigma\}$.

Intuitively, $\check{A}$ corresponds to the old value of A while $\hat{A}$ corresponds to the new value, $\check{\Sigma}$ and $\hat{\Sigma}$ are the "copies" of Σ corresponding to the two versions of the attributes, $\check{x}$ and $\hat{x}$ are "copies" of x.

With this notation we proceed with the definition of an action relation.

Definition. The action relation associated with an update (I, μ, I') is the relation[8] $I\mu I' = \{\check{x} \times \hat{\mu}(x) | x \in I\}$.

Example. Consider the update given by Fig. 3.1.

A	B		A	B
0	1	⟶	1	0
1	1	⟶	1	1

μ

Fig. 3.1

Its associated action relation is given in Fig. 3.2.

$\check{A}$	$\check{B}$	$\hat{A}$	$\hat{B}$
0	1	1	0
1	1	1	1

Fig. 3.2

The concepts we define next are of primary importance in our investigation. They represent the formalization of dynamic constraints as an analog of functional dependencies to updates.

Definition. A dynamic functional dependency (dfd) over U is an fd $X \to Y$ over $\check{U}\hat{U}$ such that, for each $A \in Y$, $XA \cap \check{U} \neq \emptyset$ and $XA \cap \hat{U} \neq \emptyset$.

Informally, the above condition on fd's $X \to Y$ over $\check{U}\hat{U}$ ensures that $X \to Y$ does not imply any non-trivial fd's over $\check{U}$ or over $\hat{U}$ (these would not truly be dynamic constraints). For example if U = ABC, then $\check{A} \to \hat{B}$ is a dfd while $\check{A} \to \hat{B}\hat{C}$ is not.

The reader will note that dfd's are a natural extension of fd's, and that other types of constraints can be extended in a similar manner. In fact, each type ϕ of static constraint gives rise to a dynamic analog $d\phi$.

We now show how to use dfd's as a constraint on updates.

Definition. An update (I, μ, I') satisfies a set Δ of dfd's, written $(I, \mu, I') \models \Delta$, if the action relation $I\mu I'$ satisfies Δ.

For example, the update in Figure 3.1 satisfies $\check{B} \to \hat{A}$ but not $\hat{A} \to \check{A}$.

Using the dynamic constraints on updates, we now place constraints on dbs's. Valid dbs's are described by a set of static fd's (Σ) and a set of dfd's (Δ). This leads to the notion of a dfd schema:

Definition. A dfd schema is a triple (U, Σ, Δ), where U is a set of attributes, Σ is a set of fd's over U and Δ is a set of dfd's over U.

We now have

Definition. Let (U, Σ, Δ) be a dfd schema. A database sequence $\{(I_i, \mu_i)\}_{0 \leq i \leq n}$ over U satisfies (Σ, Δ) if each I_i satisfies Σ, $0 \leq i \leq n$, and each update (dom μ_i, μ_i, range μ_i) satisfies Δ, $0 \leq i < n$. The set of all dbs's over U satisfying (Σ, Δ) is denoted by $Sat_U(\Sigma, \Delta)$.

Remark. Consider an update (I, μ, I') over U. Since tuples do not repeat in a relation, the action relation $I\mu I'$ does not contain distinct tuples which agree on $\check{U}$ or on $\hat{U}$. Consequently, each action relation corresponding to an update over U vacuously satisfies the dfd's $\check{U} \to \hat{U}$ and $\hat{U} \to \check{U}$. In a sense, these dfd's are trivial. However, they clearly do not follow from the traditional inference rules for fd's. In order to avoid problems arising from the "incompleteness" of the inference rules within the class of action relations, each set of dfd's over U will henceforth be assumed to contain the above two dfd's.

It turns out that arbitrary dfd's are sometimes inconvenient to deal with mathematically. A tractable subclass of the dfd's, called "bipartite dfd's", is suitable for most real

---

[8] The symbol $\times$ denotes the cross-product of two tuples over disjoint domains.

situations. (In particular, all the dfd's in Examples 2.1 and 2.2 are bipartite.) The relation between dfd's and bipartite dfd's is similar to that between phrase-structure grammars and context-free grammars. the first is appealing through its generality while the second is mathematically simpler but still powerful enough to model most applications.

Definition. A dfd $X \to Y$ over U is a bipartite dfd (bdfd) if either $X \subseteq \check{U}$ and $Y \subseteq \hat{U}$ or $X \subseteq \hat{U}$ and $Y \subseteq \check{U}$. The set of all bdfd's over U is denoted by BDFD(U). Finally, a bdfd schema is a dfd schema (U, Σ, Δ), where $\Delta \subseteq$ BDFD(U).

For instance, given U = ABC, $\check{A}\check{B} \to C$ is a bdfd, while $\check{A}\hat{B} \to \hat{C}$ is not. (Also, note that the "trivial" dfd's $\check{U} \to \hat{U}$ and $\hat{U} \to \check{U}$ are bdfd's.)

While most definitions in the paper are given for arbitrary dfd's, many of the results hold just for bdfd's.

Note that a restriction analogous to the "bipartite" restriction on dfd's can be made on the dynamic extensions of other types of static constraints, e.g., embedded implicational dependencies [F].

4. Dynamic Mappings.

In this section it is shown how to infer static constraints in the context of a single update. This is done using the notion of "dynamic mappings". Dynamic mappings describe the connection between the dynamic constraints satisfied by an update and the static constraints satisfied by the "old" and "new" versions of an instance. The four possible combinations of "old" and "new" yield four different kinds of dynamic mappings. After defining dynamic mappings we exhibit a method for computing them and give some of their basic properties. Finally, we present an important "decomposition" result which allows more efficient strategies for computing the mappings. Dynamic mappings play a prominent role in the investigations discussed in Sections 5 and 6.

We now define the four types of dynamic mappings.

Definition. Let Δ be a set of dfd's over U. The following four mappings from $2^{FD(U)}$ into $2^{FD(U)}$ are the dynamic mappings associated with Δ.

(a) The old-new mapping on_Δ associated with Δ is defined by $on_\Delta(\Sigma) = \{X \to Y \in FD(U) \mid$ for each update (I, μ, I') over U satisfying Δ, if $I \models \Sigma$ then $I' \models X \to Y\}$.

(b) The new-old mapping no_Δ associated with Δ is defined by $no_\Delta(\Sigma) = \{X \to Y \in FD(U) \mid$ for each update (I, μ, I') over U satisfying Δ, if $I' \models \Sigma$ then $I \models X \to Y\}$.

(c) The old-old mapping oo_Δ associated with Δ is defined by $oo_\Delta(\Sigma) = \{X \to Y \in FD(U) \mid$ for each update (I, μ, I') over U satisfying Δ, if $I \models \Sigma$ then $I \models X \to Y\}$.

(d) The new-new mapping nn_Δ associated with Δ is defined by $nn_\Delta(\Sigma) = \{X \to Y \in FD(U) \mid$ for each update (I, μ, I') over U satisfying Δ, if $I' \models \Sigma$ then $I' \models X \to Y\}$.

Note that dynamic mappings are analogous to logical implication. For example, for each $\Sigma \subseteq FD(U)$, $on_\Delta(\Sigma)$ is the set of all fd's which must be satisfied by each "new" instance I' over U whenever the "old" instance I satisfies Σ and the update (I, μ, I') satisfies Δ. Similar remarks can be made for the other dynamic mappings.

In [V], some basic properties of dynamic mappings are presented (omitted here). Then it is shown how dynamic mappings can be effectively computed

4.1 Proposition.[9] For each set Δ of dfd's and Σ of fd's over U,

(a) $on_\Delta(\Sigma) = \overline{\Pi_{\hat{U}}((\check{\Sigma} \cup \Delta)^*)}$,

(b) $no_\Delta(\Sigma) = \overline{\Pi_{U}((\hat{\Sigma} \cup \Delta)^*)}$,

(c) $oo_\Delta(\Sigma) = \overline{\Pi_{\check{U}}((\check{\Sigma} \cup \Delta)^*)}$, and

(d) $nn_\Delta(\Sigma) = \overline{\Pi_{\hat{U}}((\hat{\Sigma} \cup \Delta)^*)}$.

In some sense, the mappings no_Δ and nn_Δ can be viewed as symmetric to on_Δ and oo_Δ, respectively. This is formalized next.

---

[9] For each $X \subseteq \check{U}\hat{U}$, $\overline{X} = \{A \in U \mid \check{A} \in X$ or $\hat{A} \in X\}$. For each $\Gamma \subseteq FD(\check{U}\hat{U})$, $\overline{\Gamma} = \{\overline{X} \to \overline{Y} \mid X \to Y \in \Gamma\}$. Thus, the symbol $\overline{\ }$ acts as an operator which erases the markers $\check{}$ and $\hat{}$. For each $V \subseteq U$ and $\Sigma \subseteq FD(U)$, $\Pi_V(\Sigma)$ is the set of all fd's over V which are in Σ.

4.2 Proposition. (Symmetry).[10] For each set Δ of dfd's over U, $no_\Delta = on_{\sigma(\Delta)}$ and $nn_\Delta = oo_{\sigma(\Delta)}$.

The Symmetry Proposition allows us to obtain results about no_Δ and nn_Δ from corresponding results about on_Δ and oo_Δ, respectively.

We now present an important "decomposition" theorem which allows a more efficient computation of dynamic mappings in certain situations. This non-trivial result is only true for bdfd's.

4.3 Theorem. Let U be a set of attributes, Δ a set of bdfd's and Σ a set of fd's over U. Then $on_\Delta(\Sigma) = [\bigcup_{f \in \Sigma^*} on_\Delta(f)]^*$.

In view of the Symmetry Proposition, an analogous result can be proven for new-old mappings. An even stronger result holds for old-old and new-new mappings. Indeed, it can be shown that $oo_\Delta(\Sigma) = [\Sigma \cup oo_\Delta(\emptyset)]^*$, and $nn_\Delta(\Sigma) = [\Sigma \cup nn_\Delta(\emptyset)]^*$.

5. Age in a Database.

In this section we look at how information about the history of the database can be used to infer static constraints satisfied in the current state. The relevant information is the "age" of tuples in the current state, i.e. the number of updates to which a tuple was subjected since it was first inserted in the database.

Since our dynamic constraints involve only updates, it is convenient to first look at databases with no insertions or deletions, called "stable". Formally, a dbs $\{(I_i, \mu_i)\}_{0 \le 0 \le n}$ is called stable if dom $\mu_i = I_i$ (no deletions) and range $\mu_i = I_{i+1}$ (no insertions) for each i, $0 \le i < n$. The notion of age is formalized for stable dbs's as follows

Definition. Let $s = \{(I_i, \mu_i)\}_{0 \le i \le n}$ be a stable dbs and u a tuple in I_k, $0 \le k \le n$. The occurrence of u in I_k has age k in s.

We will simply say that a tuple has age k whenever it is clear which occurrence of the tuple we are referring to.

Clearly, the notion of age becomes irrelevant if each identity update of a valid instance of the database satisfies Δ. (An identity update is an update (I, μ, I') where $\mu(x) = x$ for each x in I.) Indeed, any instance of the database could then become arbitrarily "old" simply by staying unchanged. It can be shown that, given a dfd schema (U, Σ, Δ), each identity update of a valid instance satisfies Δ iff $\bar{\Delta} \subseteq \Sigma^*$ (recall that the operator - erases the markers $\check{}$ and $\hat{}$).

Consider now a database satisfying (Σ, Δ). As seen from Examples 2.1 and 2.2, if tuples attain a certain age in the database, then the set of such tuples must sometimes satisfy static constraints in addition to Σ^*. Thus, we can talk about the "closure" of Σ in the context of age information.

Definition. For each dfd schema (U, Σ, Δ) and each $k \ge 0$, the age-k closure of Σ under Δ is the set $\Sigma_k^\Delta = \{f \in FD(U) \mid$ if S is a set of tuples of age k in some stable dbs in $Sat_U(\Sigma, \Delta)$, then $S \models f\}$.

Thus, Σ_k^Δ is the set of all fd's satisfied by each set of tuples of age k in some stable dbs in $Sat_U(\Sigma, \Delta)$.

We now show how age-k closure can be computed for bdfd's, by establishing a recurrence relation between Σ_k^Δ and Σ_{k+1}^Δ (which involves the dynamic mapping on_Δ).

5.1 Theorem. Let (U, Σ, Δ) be a bdfd schema. Then $\Sigma_0^\Delta = \Sigma^*$ and

$$\Sigma_{k+1}^\Delta = [\Sigma_k^\Delta \cup on_\Delta(\Sigma_k^\Delta)]^* = [\Sigma \cup on_\Delta(\Sigma_k^\Delta)]^*,$$

for each $k \ge 0$.

We next look at the sequence $\{\Sigma_k^\Delta\}_{k \ge 0}$ of successive age-k closures of Σ under Δ. It is shown that age-k closures become constant after k has reached a certain value. Furthermore, the convergence is "rapid", in the sense that the consecutive age-k closures must increase at each step before reaching the "limit".

5.2 Theorem. For each bdfd schema (U, Σ, Δ) there exists an integer $\tilde{k}(\Sigma, \Delta) \ge 0$ and $\tilde{\Sigma}^\Delta \subseteq FD(U)$ (or $\tilde{\Sigma}$ when Δ is understood), such that $\Sigma_k^\Delta \subsetneq \Sigma_{k+1}^\Delta$ if $0 \le k < \tilde{k}(\Sigma, \Delta)$ and $\Sigma_k^\Delta = \tilde{\Sigma}^\Delta$ if $k \ge \tilde{k}(\Sigma, \Delta)$.

Clearly, $\tilde{k}(\Sigma, \Delta)$ is bounded by the number of non-trivial fd's over U not in Σ^*. However, there

---

[10] Let σ be the mapping on $\check{U}\hat{U}$ defined by $\sigma(\check{A}) = \hat{A}$ and $\sigma(\hat{A}) = \check{A}$ for each $A \in U$, σ is extended to sets of attributes and then to fd's over $\check{U}\hat{U}$ in the obvious fashion.

is no uniform bound for $\tilde{k}(\Sigma, \Delta)$. Indeed, we show that for each $n \geq 0$ there exists a bdfd schema (U, Σ, Δ) such that $\tilde{k}(\Sigma, \Delta) = n$.

Note that, in Example 2.1, $\tilde{k}(\{s_1\}, \{d_1, d_3\}) = 1$ and $\{\tilde{s}_1\} = \{s_1, s_2\}^*$. In Example 2.2, $\tilde{k}(\{s1\}, \{d1,d2\}) = 3$, and $\{\tilde{s1}\} = \{s1,s4\}^*$.

In a sense, one can say that a (stable) database becomes "mature" at age $\tilde{k}(\Sigma,\Delta)$. Indeed, no additional static constraints are acquired past that age by the database as a result of getting "older". Since $\tilde{k}(\Sigma,\Delta)$ can be any integer, it can take arbitrarily long for a database to become mature.

The set of static constraints satisfied by a mature database is $\tilde{\Sigma}$ (which may be larger than Σ^*). Satisfaction of some constraints in addition to Σ^* may itself be of use (e.g., in query processing). Also, the set $on_\Delta(\tilde{\Sigma})$ of static constraints is preserved by a valid update (i.e., an update satisfying Δ) of a mature database. In particular, the set $on_\Delta(\tilde{\Sigma}) \cap \Sigma$ is "inherited" through a valid update. Thus fewer constraints need be checked in order to make sure the new database satisfies Σ. The best one can expect is that $on_\Delta(\tilde{\Sigma}) \supseteq \Sigma$, in which case all of Σ is preserved. The worst is when $on_\Delta(\Sigma) \cap \Sigma = \emptyset^*$, in which case none of the static constraints are preserved through a valid update. Most cases fall somewhere between these extremes.

We now present an inference-rule-based mechanism to compute $\tilde{\Sigma}$ and $on_\Delta(\tilde{\Sigma})$ directly from Σ and Δ. The mechanism is, in some sense, sound and complete. The rules involve attributes marked with $\check{}$ and $\hat{}$, and consist of any sound and complete set of inference rules for fd's, together with one extra rule. The additional rule allows us to infer the fd $\check{X} \rightarrow \check{Y}$ over the '$\check{}$' attributes from the corresponding fd $\hat{X} \rightarrow \hat{Y}$ over the '$\hat{}$' attributes. This is formalized in the following

Definition. The following is called a $\tilde{\sigma}$-rule

For each $X, Y \subseteq U$, if $\hat{X} \rightarrow \hat{Y}$ then $\check{X} \rightarrow \check{Y}$.

Let $\mathcal{R}$ be a sound and complete set of inference rules for fd's over $\check{U}\hat{U}$. Let $\tilde{S}_\mathcal{R}$ be a new set of inference rules consisting of $\mathcal{R}$ together with the $\tilde{\sigma}$-rule. Clearly, the closure of a set F of fd's with respect to $\tilde{S}_\mathcal{R}$ is independent of the choice of $\mathcal{R}$.

Notation. For each $F \subseteq FD(\check{U}\hat{U})$, let F^+ denote the closure of F with respect to $\tilde{S}_\mathcal{R}$.

The computation of $\tilde{\Sigma}$ from Σ and Δ involves several steps:

1. Start with the set $F = \check{\Sigma} \cup \Delta \cup \hat{\Sigma}$,
2. Compute F^+,
3. Select all fd's over $\hat{U}$ which are in F^+, and
4. Remove all $\hat{}$ markers.

The result of the above four steps is the set $\overline{\Pi_{\hat{U}}((\check{\Sigma} \cup \Delta \cup \hat{\Sigma})^+)}$. The next theorem shows that this is equal to $\tilde{\Sigma}$.

A similar procedure can be used to compute $on_\Delta(\tilde{\Sigma})$ from Σ and Δ.

(1') Start with the set $F = \check{\Sigma} \cup \Delta$,

(2') Compute F^+,

(3') Select all fd's over $\hat{U}$ which are in F^+, and

(4') Remove all $\hat{}$ markers.

The result of (1')-(4') is the set of fd's $\overline{\Pi_{\hat{U}}((\check{\Sigma} \cup \Delta)^+)}$. As shown in the next theorem, this is equal to $on_\Delta(\tilde{\Sigma})$.

5.3 Theorem. For each bdfd schema (U, Σ, Δ),

(a) $\tilde{\Sigma} = \overline{\Pi_{\hat{U}}((\check{\Sigma} \cup \Delta \cup \hat{\Sigma})^+)}$, and

(b) $on_\Delta(\tilde{\Sigma}) = \overline{\Pi_{\hat{U}}((\check{\Sigma} \cup \Delta)^+)}$

The notion of age is extended from stable to arbitrary dbs's in the obvious way (details omitted). The closure Σ_k^Δ is the set of all fd's satisfied by each "subinstance" (of a database satisfying (Σ, Δ)), whose tuples have age k in the database.

6. Survivability in a Database.

In Section 5 we saw how to use knowledge about the history of a database to gain information about the current state. In this section we are mostly concerned with the connection between the current state and the future evolution of the database.[11]

[11] Due to space limitations, most definitions and results are stated informally, and examples are omitted.

The relevant concept here is that of "survivability" in a database. Informally, a set T of tuples over U has survivability k if the tuples can be validly updated together k times. In the case where $k = \infty$, T is said to be potentially immortal.

As in the case of age, in order for a set of tuples to have survivability k in a database, it is sometimes necessary that the set of such tuples satisfy static constraints in addition to Σ^*. This leads to the notion of survivability-k closure of Σ under Δ, denoted Σ^{Δ}_{-k}. This notion is symmetric to that of age (for k finite). The symmetry is formalized by the following result[12].

6.1 Proposition. For each bdfd schema (U, Σ, Δ)

and each $k \geq 0$, $\Sigma^{\Delta}_{-k} = \Sigma^{\sigma(\Delta)}_{k}$.

In view of the above result, we can use most of our results on age-k closure to obtain analogous results on survivability-k closure (for k finite). [In particular, the "limit" of the sequence $\{\Sigma^{\Delta}_{-k}\}_{k \geq 0}$ is denoted by $\tilde{\Sigma}^{\Delta}$.] We omit these results here.

The most notable breach in the past-future symmetry arises from the fact that, while we assume that any database has a start in time, there is no reason to assume it will not go on forever. Hence there is no analog of potential immortality (or survivability ∞) for age. Most of the results in this section focus therefore on potential immortality. We first look at the relation between the survivability-∞ closure ($\Sigma^{\Delta}_{-\infty}$) and the survivability-k closures, for k finite.

We obtain the following "convergence" result.

6.2 Theorem. For each bdfd schema (U, Σ, Δ),

$$\Sigma^{\Delta}_{-\infty} = \bigcup_{k \geq 0} \Sigma^{\Delta}_{-k} = \tilde{\Sigma}^{\Delta}.$$

Given a valid instance I of a database, it is sometimes of interest to know if I can be validly updated a given number k of times. [In fact, it may often be reasonable to require that I be potentially immortal, i.e. $k = \infty$.] Satisfaction of Σ^{Δ}_{-k} by I is a necessary condition in order that I have survivability k. However, it is not a sufficient condition (example omitted). The question arises whether it is even decidable if an instance has survivability k in a database. For k finite the problem is clearly decidable. We show that the problem is decidable also for $k = \infty$, by exhibiting a connection between finite survivability and potential immortality.

6.3 Theorem. Let (U, Σ, Δ) be a bdfd schema and I be an instance over U. If I has survivability $(2^{|U|})^{|I|^2}$ then I is potentially immortal.

In [V], a characterization is given of the bdfd schemas (U, Σ, Δ) for which every (valid) instance of the database is potentially immortal.

The remainder of the section contains, in [V], an investigation of the combined effect of age and survivability on the static constraints of a database. The main result is, intuitively, that the combined effect of age and survivability is "additive".

Acknowledgments

I am grateful to my advisor, Seymour Ginsburg, for providing guidance and support while I wrote my Ph.D. thesis, on which this paper is based. Special thanks to Richard Hull for numerous discussions and very useful suggestions related to the thesis and this paper.

SELECTED REFERENCES

[Bro] Brodie, M. On modelling behavioral semantics of databases. Proc. 7th Int. Conf. on Very Large Data Bases (1981), pp. 32-42.

[CaFu] Casanova, M.A., and A.L. Furtado. A family of temporal languages for the description of transition constraints. To appear.

[CoCaFu] Castilho, I.M.V. de, Casanova, M.A., Furtado, A.L. A temporal framework for database specifications. Proc. 8th Int. Conf. on Very Large Data Bases (1982), pp. 280-291.

[CePBra] Ceri, S., Pelagatti, G., Bracchi, G. Structured methodology for designing static and dynamic aspects of data base applications. Information Systems, Vol. 6, No. 1, pp. 31-45.

[12] Recall that σ was defined in a footnote to the Symmetry Proposition (§4).

[ClW] Clifford, J., Warren, D.S. Formal semantics for time in databases. Proc. XP2 Workshop on Relational Databases (1981).

[DeAZ] De Antonellis, V., Zonta, B. Modelling events in data base applications design. Proc. 7th Int. Conf. on Very Large Data Bases (1981), pp. 23-31.

[Da] Darwin, C. The Origin of Species (1859).

[F] Fagin, R. Horn clauses and database dependencies. Proc. ACM SIGACT Symp. on Theory of Computing (1980), pp. 123-134.

[G] Gallaire, H. Impacts of logic in databases. Proc. 7th Int. Conf. on Very Large Data Bases (1981), pp. 248-259.

[HaM] Hammer, M., McLeod, D.J. Semantic integrity in a relational data base system. Proc. 1st Int. Conf. on Very Large Data Bases (1975).

[HuYap] Hull, R., Yap, C.K. The format model: a theory of database organization. Proc. ACM Symp. on Principles of Database Systems (1982), pp. 205-211.

[NYaz] Nicolas, J.U., Yazdanian. Integrity checking in deductive data bases. From Logic and Databases (ed. by H. Gallaire and J. Minker), Plenum Press (1978).

[S] Spyratos, N. An operational approach to data bases. Proc. ACM Symp. on Principles of Database Systems (1982), pp. 212-220.

[Ul] Ullman, J. Principles of Database Systems. Computer Science Press (1980).

[V] Vianu, V., Dynamic constraints and database evolution. Ph.D. thesis, University of Southern California, January 1983.