Data Model Equivalence

by

Sheldon A. Borkin

(to be published in Proceedings of the Fourth International Conference on Very Large Data Bases, IEEE, to be held in Berlin, Germany, 13-15 September 1978)

This research was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-75-C-0661.

22 June 1978
DATA MODEL EQUIVALENCE

Sheldon A. Borkin

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts U.S.A.

Abstract

The current proliferation of proposals for database system data models and the desire for database systems which support several different data models raise many questions concerning "equivalence properties" of different data models. To answer these questions, one first needs clear definitions of the terms under discussion. This paper presents formal definitions of the terms database, operation, operation type, application model and data model.

Using this formal framework, database state equivalence, operation equivalence, application model equivalence and data model equivalence are distinguished. Three types of application and data model equivalence are defined: isomorphic, composed operation and state dependent. Possibilities for partial equivalences are mentioned. Implementation implications of these different equivalences are discussed.

Examples are presented using two semantic data models, the semantic relation data model and the semantic graph data model.

Outline

1. Introduction
   1.1 Data Models
   1.2 Multiple Data Model Database System Architecture
   1.3 Goals of this Paper

2. Formal Definition of Data Model
   2.1 Definition of Application Model
   2.2 Definition of Database and Database State

3. Formal Equivalence Definitions
   3.1 Database State Equivalence
   3.2 An Example of Equivalent Semantic Database States
      3.2.1 The Semantic Relation Data Model
      3.2.2 The Semantic Graph Data Model
   3.3 Data Model and Application Model Equivalence
      3.3.1 Application Model Equivalence
      3.3.2 Data Model Equivalence

4. Conclusions

This research was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-75-C-0681.
should take. Many suggestions are found in Nijssen. Other appropriate data models will be mentioned in this paper.

The separation of the conceptual schema from the internal schema results in data independence (Date and Hoesein). That is, the details of the physical storage representation can be altered without affecting the view of the database presented to the user. The centralized conceptual schema guarantees, so long as only "appropriate" external schemas are allowed, that every user sees the database and performs updates consistent with the application being modeled.

Finally, external schemas provide the database system user with a view of the database appropriate to his specific needs and desired data model. The schemas are to be compatible with the particular programming or query language of the user. The external schema may present to the user just a subset of the information described in the conceptual schema. While this paper will be most relevant to the case in which the external schema describes all of the data present in the conceptual schema, the definitions to be presented can be extended to handle the case where the external schema describes a subset of the conceptual schema.

This architecture requires several mapping functions - from the conceptual schema to the internal schema and from the conceptual schema to each external schema. For example, a query from a user posed in terms of an external schema must be translated into a query in terms of the conceptual schema, which must in turn be translated into a query which actually accesses the stored data. Results of the query must likewise be translated from an internal representation to a form appropriate to the user and the external schema.

The problem of how to define the mappings between schema levels so that all users see "equivalent" models of the application for both retrieval and update is extremely difficult for any case other than letting the external schema be a simple subset of the conceptual schema. Klug and Tschritz$^{20}$ provide an example of the complexities involved.

1.3 Goals of this paper

Some questions which can be raised in light of the preceding discussion are:

- How does one know if two data models are "equivalent" in expressive power so that the set of expressible external schemas is equivalent to the set of expressible conceptual schemas?

To answer these questions, one first needs clear definitions of the concepts under discussion. Section 2 of this paper provides formal definitions of the terms database, operation, operation type, application model and data model.

Section 3 uses this formal framework to distinguish between database state equivalence, operation equivalence, application model equivalence and data model equivalence. Three types of application and data model equivalence will be defined - isomorphic, composed operation and state dependent. Possibilities for partial equivalences are mentioned.

It is claimed in Section 3 that the key definition of database state equivalence can be done most convincingly when both data models of concern are semantic data models. An example is presented.

Section 4 presents conclusions.

2. Formal Definition of Data Model

A basic premise of what follows is that the database system is used to model some portion of the "real world" which is of interest to the system user. We will call this portion of the real world the application. The application state represents a "snapshot" of the application at a given time.

The database state may consist of objects which are in a 1:1 correspondence with the application state (a common interpretation of the network approach) or it may consist of "statements" about the application state (an interpretation of the relational approach). Other approaches are possible. In any case, we assume that the database models some application which can be thought of as having a state and certain allowed transitions between states. In defining equivalences in this paper we will be concerned about different data models' capabilities in expressing these application models, their states and their transitions.

A data model is then defined as a (finite or infinite) set of application models:

\[
\text{data model} \rightarrow \{\text{application model}_1, \text{application model}_2, \ldots, \text{application model}_n\}
\]

Each of the application models specifies some application or view of an application to the database system.

2.1 Definition of Application Model

An application is represented in the database system by an application model. To specify the allowed states and transitions for each application, an application model consists of a schema and a finite set of operation types:

\[
\text{application model} \rightarrow (\text{schema}, \text{operation type}_1, \text{operation type}_2, \ldots, \text{operation type}_m)
\]

The schema corresponds to the usual notion of database schema. For the relational data model it would specify the name of each relation, the domains of allowed values for each column of a relation and the integrity constraints to be satisfied by the tuples in the relation. Codd$^{10}$ calls this the "intension" of the relations. In the DRTO data model the schema would specify the various record and set types. The
schema can be considered to be the "declarative" portion of the application model.

Each operation type is a function

operation type : (schema × arguments) → database state.

Given a schema, a database state and some arguments whose values depend on the specific operation type, an operation type defines a new database state. One such possible new state is the error state. We will not be concerned here with retrieval operations which can be thought of as simply displaying some subset of the database state.

Operation types correspond to the relational model to insert tuples and delete tuples. DBTG operation types would be store, delete, rename and modify. The operation types are the "procedural" portion of the application model. Note that we consider the application model itself to be static. Clearly, at some "higher level" there must be some means of declaring and modifying the application model.

We use an application model rather than a schema and will speak of application model equivalence rather than schema equivalence to emphasize the importance of allowed operations in defining a data model. Also note that the formal view here allows different application models in the same data model to have different sets of operation types. This would be necessary for data models based on the notion of abstract data types (Liskov et al. and Smith and Smith).

We will distinguish between operation types and operations. An operation is a function which maps each possible database state into another database state.

operation : database state → database state

Given a schema and the set of possible other arguments for each operation type, we can generate an application model's set of allowable operations. For example, given a relational schema, there would be an operation corresponding to the insertion or deletion of each possible set of tuples.

2.2 Definition of Database and Database State

Thus far we have used the term database state without any further explanation. The specific form of the database state depends on the data model under discussion. In the relational model, the state would be a mapping from relation names to sets of tuples. In Codd's terminology, this is the extension of the relations. A DBTG state would consist of sets of records and indicators of set membership links.

Since the only database states of concern are those which can be reached by the set of allowed operations, we define the set of valid database states as consisting of some (initial) state, most likely the empty state, and those states consisting of the closure of the application model's set of allowable operations applied to this initial state. Each of these states is meant to represent some possible application state or else the error state.

Finally, we can define a database as specifying for an application both the current state and all possible state transitions.

```
database : (application model, database state)
```

That is, in the context of a database system there is a static component, the application model, and a dynamic component, the database state. The application model determines the set of valid operations while the database state is the database system's representation of the application state.

The definitions used to define data model are summarized in Figure 2.

3. Formal Equivalence Definitions

Now that the components of a data model have been defined, we can examine the various equivalences of interest. The notion of data model equivalence is based on application model equivalence which is in turn based on database state equivalence. We will examine these in a "bottom-up" manner.

3.1 Database State Equivalence

We would like to say that two database states expressed in terms of two different data models are equivalent if they represent identical application states. However, the precise specification of state equivalence depends on the nature of the two data models. That is, what is the interpretation of the database state in terms of the application?

The case with which such interpretations can be given varies widely between data models. Such problems of interpretation led Codd7 to the development of normal forms for the relational model. The significance and the limitations of these normal forms are explained by Schmid and Swenson30 in terms of a data model with more clearly defined semantics. We will call such data models semantic data models. A semantic data model, rather than allowing in the database state arbitrary syntactic structures such as trees, networks or relations, attempts to provide users of the data model with a clear interpretation of the database in terms of the relevant application. Some data models which we would classify as semantic data models are Chen,25 Dheeneffe et al., Hall et al.15 and Schmid and Swenson.

Most of the proposals for the ANSI conceptual schema are based on similar notions of semantics (Nijssen28). We will call other data models, including Codd's relational model and the DBTG model, syntactic data models. Bachman and Days2 have recently proposed an extension of the DBTG model which contains notions of meaning and semantics. Note that we do not claim that there is a perfectly well-defined boundary between syntactic and semantic data models. In one sense, all formally defined data models are syntactic since formal systems are in the end pure syntax. In the sense we wish to use the term, we regard a semantic data model as one which is conducive to the user having a direct interpretation of the database in terms of the application.

The difficulty of defining equivalent database states for syntactic data models can be seen by examining previous work on equivalence problems for the relational and DBTG (or similar) models. Zimmermann25 and Fleck28 both require that

---

**Figure 2 - Summary of Terms**

data model : application model, application model
application model : schema, operation type
operation type : database state → database state
database state : database state
there be a relational tuple for each DBTG record plus a binary relational tuple for each DBTG set-ownership-membership link. These restrictions on the form of the relational state, and hence schema, severely limit the types of information which a user might desire to appear together in a single relation.

Kaplan13 allows more general relations, but allows updates to be performed only on those relations whose tuples are in a bi-continuous function with the DBTG records and links. The earlier works of Neilsen25 and McGee22 allow much more general "equivalent" mappings, but ignore the problems of performing equivalent updates. King and Tschirhart20 and Paolini and Pelagatti24, who also allow general mappings point out the difficulties of performing equivalent updates for arbitrary conceptual-to-external database state mappings.

Navathe and Schkolnick24 define a data model to facilitate "view integration". View integration is the process of developing a single model of the application consistent with each user's view of the application. This is similar to the problem of ensuring that each ANSI external schema is compatible with the conceptual schema. Navathe and Schkolnick properly emphasize the importance of stating and taking into account the effects of operations altering the database state.

### 3.2 An Example of Equivalent Semantic Database States

The use of semantic data models can ease the task of defining equivalent database states. This section will show what it means for two database states, based on semantic data models substantially different in their means of representing information, to be equivalent. Bohill3 uses the framework presented in this paper to examine the equivalence properties of two data models, the semantic relation data model and the semantic graph data model, which can be thought of as "semantic versions" of Codd's relational and the DBTG data models, respectively.

We are not attempting to present in this paper complete descriptions of these data models. Rather, we are presenting just enough detail to allow the illustration of our notions of equivalence. Furthermore, it is felt that the concepts demonstrated by examining the equivalence of these data models are generally applicable to all of the other data models mentioned in this paper.

The design of both these semantic data models are influenced by the notion of case grammars (Fillmore8 and Bruce3). This claims that the "underlying" meaning of natural language sentences can be expressed as a verb phrase, a predicate, plus several noun phrases - one for each case required by the predicate. Case grammars have formed the basis for knowledge representation for use in artificial intelligence systems (Hawkinson17, Simmons22 and Mylepoulos et al.25). We propose the use of case grammars to allow the easier organization and understandability of non-inferential database systems.

#### 3.2.1 The Semantic Relation Data Model

Figure 3 shows a semantic relation database state describing a machine shop. The state consists of three relations which are sets of statements (tuples). Each relation contains the set of all true statements fitting a certain form. For example, each row in the Operate relation represents a statement of the form:

- "There is a machine of type ___ with number ___ and this machine is operated by an employee named ___.

Each row in the Employees relation represents a statement of the form:

- "There is an employee named ___ whose age is ___."

The first row of the Jobs relation states:

- "An employee named C.Gershag is supervised by an employee named G.Wayshum and operates a machine numbered JCL887."

A null value is allowed so that the second row of Jobs states:

- "An employee named T.Manhart has no supervisor and operates a machine numbered N7745."

The relation Jobs contains statements specifying the agent and object cases of the predicates supervise and operate.

The first row in a relation's heading specifies sets of predicate : case pairs. The second row specifies case types. The third row specifies characteristics while the fourth row specifies domains. The schema must contain a specification of the values comprising each domain. This data model is similar in some respects to Chen5.

The operations allowed in the semantic relation data model are the insertion and deletion of sets of statements. In addition, the database state resulting from every successful application of one of these operations is guaranteed to satisfy a set of constraints specified as part of the schema.
We will not show the constraints for this example, but they could include such conditions as:

1. The names in the first column of Operate must be a subset of the names in the first column of Employees.
2. The first column of Operate may have no null values since every machine must have an operator.
3. A specific serial number may occur only once in the second column of Operate since each machine may have no more than one operator.
4. The matching of operators and machines occurring in Operate must be the same as that in Jobs.

The constraints, as fully described in Barkin, are semantic counterparts of functional dependencies, subset constraints and other such constraints as used in the syntactic relational model (Codd, Stonebraker, and Hammer and McLeod). Some of the constraints are expressed in terms of semantic counterparts of syntactic relational algebra operations such as projection and join.

For example, to reflect the semantics of the relations, three distinct operations, case-join, predicate-join and conjunction, replace the syntactic join. Case-join combines two relations describing different characteristics of the same predicate-case pair into a single relation. Predicate-join combines two relations describing different cases of the same predicate into a single relation. Conjunction combines two relations containing different predicates into a single relation.

3.2.2 The Semantic Graph Data Model. While the semantic relation database state contains statements about the application state, the semantic graph state is meant to consist of objects in 1:1 correspondence with the application state. This data model is similar to those of Schmid and Swenson and Delboeuf et al. Figure 4 shows a database state in the semantic graph model which is equivalent to the semantic relation state of Figure 3.

The database state consists of entities, associations and characteristics joined by role and characteristic edges. Associations correspond to the "events" in an application state which are described by some predicate. Roles correspond to the case grammar notion of cases. In Figure 4, there are two types of entities, employees and machines, and two types of associations, operation and supervision.

The operations in this data model are meant to directly model the kinds of transitions which can take place in the application. The operations allowed are the insertion or deletion of an independent entity, an independent association or a semantic unit. A semantic unit is a group of entities and associations which must be inserted or deleted as a single unit due to restrictions stated in the schema.

The schema corresponding to Figure 4 is shown in Figure 5. This schema states, via the distinction between solid and dotted edges, that every machine must be part of an operation association but not every employee need be in an operation association. The arrowheads in the schema specify functionality constraints. For example, the arrowheads state that employees are uniquely identified by their name while the identity of both the agent and object roles are necessary to uniquely identify a supervision association. They also state that a machine may belong to only one operation association. Therefore, a semantic unit is formed from a machine and its associated operation association. Whenever a machine is inserted or deleted, an operation association must also be inserted or deleted.

3.2.3 Defining Database State Equivalence. We have stated that Figures 3 and 4 represent equivalent states. We could show this by translating each relational statement into a formal logic statement and then showing that the semantic graph state is a model in formal logic sense for the set of logical statements. A prerequisite to this would be establishing between the semantics of the two data models. For example, a mapping between association types and the predicate used to express information concerning each association type would be required ("supervision" and "operate"). That is, there must be a translation between the natural language case grammars on which the two data models are based. Hopefully, the natural language meaning of the words would help confirm that the mappings are done correctly.

All of the formalisms presented in this paper are also relevant for maintaining equivalence between the conceptual and internal levels. However, here it is difficult to use only semantics to define database equivalence since the internal schema presumably contains much implementation information which has no equivalent at the conceptual level. The corresponding state and operation mappings may be very complex.
3.3 Data Model and Application Model Equivalence

Now we will return to the problem of defining data model equivalence for any two data models. Recall that a data model is a set of application models. We will consider two data models

\[ M = \{ m_{\text{model}}_1, \ldots, m_{\text{model}}_m \} \]

and

\[ N = \{ n_{\text{model}}_1, \ldots, n_{\text{model}}_n \}. \]

Our goal is then to specify under what conditions we can say that \( M \) is equivalent to \( N \).

3.3.1 Application Model Equivalence

Next we want to define application model equivalence. That is, how to identify when application models from different data models represent the same application. For the ANSI architecture, we would want to show that the internal application model, the conceptual application model and each external application model are equivalent.

We will consider defining application model equivalence for two application models \( m_{\text{models}} M \) and \( m_{\text{models}} N \). We will assume that \( M \)-ops and \( N \)-ops, and \( M \)-states and \( N \)-states define the sets of allowable operations and the sets of valid states for these two application models. We will refer to database states and operations from these sets as \( m_{\text{op}} \) and \( m_{\text{op}} \), and \( n_{\text{state}} \) and \( n_{\text{state}} \), respectively.

We will assume that the state equivalence correspondence has been defined between \( M \)-states and \( N \)-states. That is, given states \( m_{\text{state}} \) and \( n_{\text{state}} \), we can tell whether or not they represent the same application state. We require that the state equivalence correspondence be onto both \( M \)-states and \( N \)-states. That is, the "expressive power" of the state representation schemes are equal in both application models. It also seems reasonable to assume that for both application models, some specific state of application is represented by a unique state. Hence, we require state equivalence to be a 1-1 onto correspondence. This does not require that there be a mapping between components of equivalent states as required, for example, by the mapping of each relational tuple to either a DUTG record or a set membership link in Zimmermann. The specific manner of defining the state equivalence relation is dependent upon the data model being considered.

A straightforward approach to defining application model equivalence is to require a 1-1 onto correspondence, operation equivalence, between the sets of operations \( M \)-ops and \( N \)-ops such that state equivalence and operation equivalence form an isomorphism. We make the assumption that the error states of all application models are equivalent.

First we define operation equivalence.

Definition 2: A function

\[ m: M_{\text{states}} \rightarrow M_{\text{states}} \]

is operation equivalent to a function

\[ n: N_{\text{states}} \rightarrow N_{\text{states}} \]

if and only if for any two equivalent states \( m_{\text{state}} M_{\text{states}} \) and \( n_{\text{state}} N_{\text{states}} \), \( m(m_{\text{state}}) \) is state equivalent to \( n(n_{\text{state}}) \).

Then define isomorphic application model equivalence.

Definition 3: Two application models \( m_{\text{model}} \) and \( n_{\text{model}} \) are isomorphically equivalent if and only if

1. For every operation \( m_{\text{op}} M_{\text{ops}} \), there exists exactly one \( n_{\text{op}} N_{\text{ops}} \) which is operation equivalent to \( m_{\text{op}} \).

and

2. For every operation \( n_{\text{op}} N_{\text{ops}} \), there exists exactly one \( m_{\text{op}} M_{\text{ops}} \) which is operation equivalent to \( n_{\text{op}} \).

This is the "most strict" form of application model equivalence we will define. However, it may be too strict. Some application models which we would like to call equivalent do not satisfy the definition. In particular, consider the semantic relation and semantic graph data models. Suppose that in the semantic relation data model we consider an operation which inserts tuples corresponding to the insertion in the graph model of independent entities. That is, the relational operation is equivalent to the composition of several graph operations.

To account for this form of equivalence, we allow the definition of operation equivalence to be applied to the sets \( M_{\text{op}} \) and \( N_{\text{op}} \) which represent the set of all compositions of operations in the two application models. This allows the definition of complex operation application model equivalence.

Definition 3: Two application models \( m_{\text{model}} \) and \( n_{\text{model}} \) are composed operation equivalent if and only if:

1. For every operation \( m_{\text{op}} M_{\text{ops}} \), there exists an operation \( m_{\text{op}} N_{\text{ops}} \) which is operation equivalent to \( m_{\text{op}} \).

and

2. For every operation \( n_{\text{op}} N_{\text{ops}} \), there exists an operation \( m_{\text{op}} M_{\text{ops}} \) which is operation equivalent to \( n_{\text{op}} \).

Definition 3 does not restrict the operation equivalence correspondence to be 1-1. It does require that there be an operation equivalent to each of the simple (i.e., non-composed) operations of \( M_{\text{ops}} \) and \( N_{\text{ops}} \). By composition of simple operations, the correspondence is onto \( M_{\text{ops}} \) and \( N_{\text{ops}} \).

In practical terms, we would hope that the operation equivalence mappings can be expressed as an algorithm rather than an explicit enumeration of an extremely large number of equivalent pairs. It is such an algorithm which would actually allow the implementation of a database system which provides users of two different data models with access to the "same" data.

Note that based on the types of operation equivalence introduced so far, the translation of operations from one application model to an equivalent application model can be done independently of the database state. That is, operations are equivalent for all possible states. Consequently, such a translation could be done at compile-time.

This is not necessarily always the case. The mappings of equivalent operations may be dependent upon the database state. Consider the case of adding to the graph database state of Figure 4 a supervision association between C.Wayman and T.Manhart resulting in Figure 6. The equivalent semantic relation database state has the same Employees and Operate relations shown in Figure 3 with the new Jobs relation shown in Figure 7.

The semantic relation operation equivalent to the stated graph operation consists of the insertion of the second tuple in the Jobs relation of Figure 7. There are two important things to notice about this operation.

- The
Firstly, we did not explicitly delete the second tuple of the
Figure 3 "jobs" relation. This is because the semantic relation
Insert-tuples operation type is defined to automatically delete
all tuples in a relation 'less than' those inserted. The partial
ordering of tuples is based on all non-null domain values
being greater than null ("---") and incomparable with any
values other than null and itself.

More importantly, the values in the added tuple are
dependent upon the database state of Figure 3. Suppose that
the semantic graph state of Figure 4 had no operation
association involving T.Manhart. This would change the
operation needed to insert the supervision association
between T.Manhart and G.Wayburn.

In the case of the relational model, the lack of an operation
association for T.Manhart would change which tuple needed to
be added to reflect the new supervision. The tuple to be
added would be the second tuple in Figure 8 rather than the
second tuple in Figure 7. King and Tschantz have also
pointed out the possibility of the state affecting the definition
of equivalent operations.

We reflect the possibility of state dependence in the
following definitions of state dependent operation and
application model equivalences:

Definition 4: A function
m : M-states \rightarrow M-states
and a function
n : N-states \rightarrow N-states
are state dependent operation equivalent for a given pair of
equivalent states m-state \& n-state, if and only if m-state = n-state.

Definition 5: Two application models m-model and n-model are
state dependent equivalent if and only if for every pair of
equivalent states m-state \& n-state:

i). For every operation m-op \rightarrow m-op', there exists an
operation n-op \rightarrow n-op', which is state dependent operation
equivalent to m-op for m-state and n-state.

and ii). For every operation m-op \rightarrow m-op', there exists an
operation n-op \rightarrow n-op', which is state dependent operation
equivalent to n-op for m-state and n-state.

The types of application model equivalence defined are
decreasingly strict. That is, isomorphic equivalence implies
composed operation equivalence, and composed operation
equivalence implies state dependent equivalence. There are
other possibilities - we could define state dependent
equivalence which does not allow composed operations. The
intuition guiding the definitions presented here is that as data
models become more dissimilar, they do so in several ways
simultaneously. This seems justified by the examples
considered.

5.3.2 Data Model Equivalence

Finally we reach the goal of defining data model equivalence.

Definition 6: Two data models M and N are (isomorphically,
composed operation, state dependent) equivalent if and only if
(isomorphic, composed operation, state dependent)
application model equivalence defines a correspondence onto
both M and N.

That is, if two data models are equivalent, then for any
application model in one data model there is an equivalent
application model in the other data model. The expressive
power of the data models are equivalent. The definition is
dependent upon the type of application model equivalence
required.

For the ANSI architecture, equivalence of the conceptual
and external data models would mean that any conceptual
application model (schema) could be viewed through the
external data model, and any external application model would
have a corresponding conceptual model.

The general semantic relation and the semantic graph data
models are not equivalent. This is due to the great freedom in
the relational model, as in some synthetic relational models,
allowed in specifying constraints. As pointed out by Borkin, there
are relational application models which do not have an
equivalent graph application model. A relational application model might be viewed as a few constraints be equivalent to a graph model. We can then say that these data models are partially equivalent. By restricting the allowed constraints, total state dependent equivalence can be defined for the semantic relation and graph data models.

Note that the definitions do not require that application model equivalence be 1-1. In the case of the semantic relation and graph data models, there may be several relational application models state dependent equivalent to each graph model. This corresponds to the many different ways of grouping into relations the statements corresponding to a single graph state. For example, Figure 9 (with appropriate constraints) shows a single semantic relation which is application model equivalent and state equivalent to the three semantic relations in Figure 3. Such a property makes the semantic relation model a good candidate for the ANSI external data model allowing many different relational views of a single semantic graph conceptual application model.

4. Conclusions

A formal framework for defining data models has been presented. Using this framework, several different types of data model equivalence were defined. This framework and these equivalence definitions should be used to evaluate the suitability of data models for the different ANSI schema levels. Insight can be gained into both the data models themselves and into the type of implementation needed.

The considerations involved in actually implementing an ANSI-like architecture must include an examination of the equivalence issues discussed in this paper. Whether the defined equivalences are state dependent or not would have a large effect on the types of implementations allowable. The operation equivalence mappings from the external to internal levels should be analyzed as a guide for implementation efficiency considerations. The same types of equivalence mappings must be involved in the transformation of a database application program from one database system to another.

Note that the framework is applicable to syntactic data models as well as semantic data models. We have simply pointed out that the task of comparing data models is easier when the data models of concern attempt to provide a clear interpretation of how they represent that portion of the real world which is of interest to the user.

For example, the studies in Boekink's defining the equivalence of the semantic relation and graph data models shed light on the syntactic relation vs. DBTG network "controversy". As in the syntactic case, the semantic relation model presents much simpler structures and operations than the graph model. However, when defining a relational application model equivalent to some given graph model, it is necessary to define a large number of possibly complex constraints making updating difficult to understand.

One way or another, the complexity of the application must be represented in the application model. The choice of data models can significantly affect where the complexity is to reside. However, the ability to support equivalent relational and graph application models accessing a shared database would allow the best of both worlds: a simple relational view for retrieval and a graph model for updating.

There should be more development of semantic data models such as the semantic relation model for the external schema level. While it may be necessary to support existing syntactic data models at the external level, all users should not be deprived of important semantic information which helps them understand how the application is modelled. User interfaces and languages should be developed for such purposes.

References


