A SET-THEORETIC DATA MODEL FOR EVOLVING DATABASE ENVIRONMENTS

E. J. Yannakoudakis, P. K. Andrikopoulos
Athens Univ. of Economics and Business, Department of Informatics
Athens 10434, Greece
{eyan, padrik}@aueb.gr

ABSTRACT

The paper presents an integrated set-theoretic data model that offers a framework for defining a unified schema for any database environment. We utilise the concepts "entity" in its classical meaning, "tag" as a set of properties (attributes) which can describe an entity, "subtag" as a set of simple atomic attributes which cannot be decomposed further, "domain" as a set of well-defined values that can be derived from pre-specified data types, "language", "vocabulary" and "message" as strings or coded values that represent human languages and corresponding messages. The model described can manage efficiently changes that occur at the logical level and supports operations and functions that offer solutions to well-known problems faced by database designers and programmers alike. Typical problems solved include the retention of multiple schema versions, the maintenance of authority files, the support of repeatable attributes, the processing of multilingual databases (at both data and interface levels).

Keywords: data models, databases, schema evolution

1 INTRODUCTION

The evolution of database schemata is becoming a very serious problem, especially with the advent of large distributed databases. Continuous modifications of a database schema are necessary due to (a) the fact that the applications are continuously changing, (b) the perspective experience from using a system that induces changes to desired functionality and (c) the scale of many tasks that require incremental design. Sjoberg addressed the problem in practice by using a method for quantifying schema changes and proposed a tool that automates modification procedures [1]. Nowadays, the problems of schema evolution and schema versioning still form common issues of research in the context of database applications which are destined to have a long lifetime.

According to a widely accepted terminology, schema evolution is accommodated when a database system facilitates the modification of the database schema without loss of existing data, whereas schema versioning is accommodated when a database system allows the accessing of all data, both retrospectively and prospectively, through user definable version interfaces [2]. Schema evolution implies that extant data will be converted from the old to the new schema. Thus, all valid instances of the old database schema will become valid instances of the new database schema and new versions of the software can access old data [3]. Despite supporting backward compatibility schema evolution fails to form an effective technique because data conversion is feasible when the scale is small and when the changes are simple. As the scale of the application increases the problem becomes more complicated and confusing.

In schema versioning the old schema and its corresponding data are preserved, but a new version of the schema is created, which incorporates all the desired changes [4]. The different versions of the schema can be identified and selected by a suitable labelling system, such as symbolic naming, or time stamping (transaction time of schema changes).

Despite attaining continued support of legacy applications, the accommodation of schema versioning presents various open problems related to the update of data through historical schemata [3]. Organizing all schema versions that arise from previous evolutions in order to cope with how applications can access the data in a versioned schema environment, besides the considerable time and effort required for implementation [5], gives rise to high computational costs.

In relational database systems the main problem
The definition of the basic elements of our model is based on the mathematical theory of (unordered) sets.

A set is a collection of objects; order is not taken into account [14]. The elements making up a set are assumed to be distinct. Given a description of a set $X$ and an element $x$, we can determine whether or not $x$ belongs to $X$. If $x$ is in the set $X$, we write $x \in X$, and if $x$ is not in $X$, we write $x \notin X$.

Every set is a subset of the set $U$, called a universal set or universe, which must be explicitly given or inferred from the context. In the definitions that follow we attempt to define every universal set, so that it can be used as a frame of reference to every set of our model.

### 2.1 Languages, Vocabulary Items and Messages

Let us define $U_L$ as the universal set of all spoken languages supported by the Unicode standard and $U_V$ the universal set of all words and phrases supported by any Unicode writing system. Certainly there is a meaning behind every element of the $U_V$ set. That helps us define $U_M$ as the universal set of the semantics of the $U_V$ set. This consideration is the first step towards the implementation of multilingual operations, while adopting the following definitions:

- Set of Languages $L \subseteq U_L$ is the unordered set of all registered languages. That is, all the languages which are currently supported.
- Set of Vocabulary Items $V \subseteq U_V$ is the unordered set of registered words and phrases, which can be used for naming data elements in a specific language $l \in L$.
- Set of Messages $M \subseteq U_M$ is the unordered set of the semantics of the $V$ element and can be used to assign conceptual meanings to data elements. That is, to name data elements in a language of the universe of discourse.

According to the above definitions a language $l$ is currently supported if $l \in L$ or is not supported if $l \in \{x \in U_L | \neg (x \in L)\}$. Similarly, a message $m \in M$ is a registered message that can be used and $v \in V$ is a registered vocabulary item that can also be used.

Matching vocabulary items with a message is performed by the $\text{Smsg}$ function. Functions are used extensively in discrete mathematics to assign to each member of a set $X$ exactly one member of a set $Y$. The binary function $\text{Smsg}$ from $D_{\text{Smsg}}$ to $V$ is a relation that each element of $D_{\text{Smsg}} \subseteq M \times L$ is associated with a unique element of $V$. The $\text{Smsg}$ function can be used to translate any message into any supported language.

### 2.2 Domains

We define $U_D$ as the universal set of every data domain that can be defined for storing information. Based on the above:

- Set of domains $D \subseteq U_D$ is the unordered set of registered domains, which determine all possible values of every data element in our model.
If a domain \( d \) is defined, then \( d \in D \) holds and \( d \) can be assigned to a data element. A domain \( d \in D \) specifies a primitive datatype value and a length value. These properties are defined by the functions Dtype and Dlen. The function Dtype from \( D_{\text{Dtype}} = D \) to \( Q = \{ \text{int}, \text{char}, \text{real}, \text{bool}, \text{blob} \} \) is a relation whereby each element of \( D \) is associated with a unique element of \( Q \). The function Dlen from \( D_{\text{Dlen}} = D \) to \( \mathbb{N} \) is a relation whereby each element of \( D \) is associated with a natural number which determines the data length in bytes.

### 2.3 Entities

We define \( U_E \) as the universal set of distinct and autonomous objects, forming what we generally call "entity". An entity need not have a material existence. In particular, abstractions and legal functions are usually regarded as entities. Based on this we define:

- **Set of Entities** \( E \subseteq U_E \) is the unordered set of registered entities that participate in the logical schema of our model.

For every entity \( e \) that has been created and is current in the logical schema the expression \( e \in E \) holds. Also, every entity in the logical schema has a unique name. The function Enam from \( D_{\text{Enam}}=E \) to \( M \) is a relation whereby each element of \( E \) is associated with a unique element of \( M \), naming in effect each entity. Note that Enam is an one-to-one function so it has an inverse function \( E^{-1} \).

### 2.4 Tags

We define \( U_T \) as the universal set of all particular properties (attributes) that can feature any entity of the model. Based on this:

- **Set of Tags** \( T \subseteq U_T \) is the unordered set of properties that exist in our logical schema and feature its entities.

Tags can be simple or complex. For every tag \( t \) that has been created and currently exists in the logical schema the expression \( t \in T \) holds.

Every tag resides in (at most) one entity. Entity-tag associations are performed by the Tent function. This function from \( D_{\text{Tent}}=T \) to \( E \) is a relation whereby each element of \( T \) is associated with a unique element of \( E \).

Every tag has a name assigned through the Tnam function. The function Tnam from \( D_{\text{Tnam}}=T \) to \( M \) is a relation whereby each element of \( T \) is associated with a unique element of \( M \).

Note that the Tnam function is not one-to-one, because two different tags may have the same name. However, a tag’s name has to be different from any other tag’s name in the same entity.

Every simple tag \( t \in T \) holds data of the same kind, which are singularly associated with every instance of the entity \( e = \text{Tent}(t) \) where tag \( t \) resides. Although data storage and manipulation are not dissertated in this model description that emphasizes on introducing the logical structure by defining metadata, it is important to note that tags can only receive values from a specific domain. The function Tdom from \( D_{\text{Tdom}} \subseteq T \) to \( D \) is a relation whereby each element of \( T \) is associated with a unique element of \( D \). A complex tag has no domain, because its domain is specified by the domains of its subtags.

Every simple tag has an occurrence status. It can either be optional or mandatory. A mandatory tag must always contain a value that is not null. This feature of a tag is defined by the Tocc function. The function Tocc from \( D_{\text{Tocc}} \subseteq T \) to \( O = \{ 0, 1 \} : \text{Optional}, 1 : \text{Mandatory} \) is a relation whereby each element of \( T \) is associated with a unique element of \( O \). A complex tag has no occurrence status, as it has no domain.

Every tag has a repetition status which can be either single-valued or multi-valued. A single-valued tag must always contain a single value for a given sample of an entity \( e \). However, multi-valued tags can receive multiple values for the same sample of an entity. The repetition status of a tag is defined by the Trep function. The function Trep from \( D_{\text{Trep}} = T \) to \( R = \{ 0, 1 \} : \text{Single} \text{−} \text{value}, 1 : \text{Multi} \text{−} \text{value} \) is a relation whereby each element of \( T \) is associated with a unique element of \( R \).

Tags from different entities may reference one another, provided they receive values from the same domain. Every tag has an authority status that determines its participation in a relation. The authority status of a tag is defined by the Taust function. The function Taust from \( D_{\text{Taust}} = T \) to \( A = \{ 0, 1, 2 \} : \text{No-participation}, 1 : \text{Authority} \text{−} \text{Tag}, 2 : \text{Selected} \text{−} \text{Tag} \) is a relation whereby each element of \( T \) is associated with a unique element of \( A \). References between tags are performed by the Auth function from \( D_{\text{Auth}} \subseteq T \) to \( T = \{ t | \text{Taust}(t) = 1, t \in T \} \). For example, let us assume that tag \( t_{A1} \) of a given entity \( A \) references tag \( t_{B1} \) of entity \( B \) and tags \( t_{B2}, t_{B3} \) from \( B \) are selected for presentation. That is, \( \text{Auth}(t_{A1}) = t_{B1} \) where \( \text{Taust}(t_{B1}) = 1 \) and \( \text{Taust}(t_{B2}) = \text{Taust}(t_{B3}) = 2 \).

### 2.5 Subtags

We define \( U_S \) as the universal set of all possible simple (atomic) attributes that can constitute any complex tag.

- **Set of Subtags** \( S \subseteq U_S \) is the unordered set of registered attributes that exist in our logical schema, which constitute existing complex tags.

For every subtag \( s \) that has been created and currently resides in the logical schema the expression \( s \in S \) holds.

Every subtag is part of one and only tag. This association is performed by the Stag Function. The function Stag from \( D_{\text{Stag}} = S \) to \( T \) is a relation whereby each element of \( S \) is associated with a unique element of \( T \).

Every subtag necessarily has a name. The naming of subtags is performed by the Snam function.
The function \( S\text{nam} \) from \( D\text{nam} \) = \( S \) to \( M \) is a relation whereby each element of \( S \) is associated with a unique element of \( M \).

The \( S\text{nam} \) function is not one-to-one, because two different subtags may have the same name. However, a subtag’s name has to be different from any other subtag’s name in the same tag.

Subtags can only receive values from a specific domain, similarly to tags. The function \( S\text{dom} \) from \( D\text{dom} \) = \( S \) to \( D \) is a relation whereby each element of \( S \) is associated with a unique element of \( D \).

Every subtag has also an occurrence status. It can either be optional or mandatory, and is defined by the \( S\text{occ} \) function. This function from \( D\text{occ} \) = \( S \) to \( O \) = \{ 0, 1 | 0 : \text{Optional}, 1 : \text{Mandatory} \} \} is a relation whereby each element of \( S \) is associated with a unique element of \( O \).

Every subtag has also a repetition status. It can either be single-valued or multi-valued, and is defined by the \( S\text{rep} \) function. This function from \( D\text{rep} \) = \( S \) to \( R \) = \{ 0, 1 | 0 : \text{Single – valued}, 1 : \text{Multi – valued} \} \} is a relation whereby each element of \( S \) is associated with a unique element of \( R \).

3 OPERATIONS

The creation and management of the structure defined in the previous section is based on operations that perform metadata manipulation. These operations affect the basic sets and the functions, as well as domains and values.

Generally, the types of operations are add, delete, rename, update and select. Add operation adds a new set element. Delete operation deletes an element from a set. Rename operation changes the alphanumerical code or the abbreviation used for an element notation. Update operation is used in more complex sets (e.g. Tags) for updating any of their features. Finally, select operation enable us to select a set element.

In what follows we present the algorithm of some typical operations.

3.1 Languages, Vocabulary Items and Messages

3.1.1 Add language

This function inserts a new element \( l \) in Languages set \( L \). If addition succeeds the function returns the modified \( L \); otherwise, it returns the empty set.

\[
\text{AddLanguage}(l)
\]

\[
x := \emptyset
\]

if \( \{l\} \cap L = \emptyset \) then

\[
L := L \cup \{l\}
\]

\[
x := L
\]

else

\[
\text{display("language } l \text{ exists")}
\]

end if

return(\( x \))

3.1.2 Select vocabulary item

This function enables the user to select an element \( v \) from Vocabulary Items set \( V \). If a selection is made, the function returns the element \( v \); otherwise, it returns the empty set.

\[
\text{SelectVocItem}()
\]

\[
x := \emptyset
\]

\[
V := V
\]

do \( \forall v \in V \)

\[
\text{display}(v)
\]

\[
\text{read}(\text{choice})
\]

if \( \text{choice} = "\text{select}" \) then

\[
x := \{v\}
\]

\[
V := \emptyset
\]

end if

end do

return(\( x \))

3.1.3 Update message

This function updates message properties. It enables the manipulation of the vocabulary items of a message in every language. If an update is effected, the function returns the updated message \( m \); otherwise, it returns the empty set.

\[
\text{UpdateMessage}(m)
\]

\[
x := \emptyset
\]

do \( \forall l \in L \)

\[
\text{read}(\text{choice})
\]

case choice of

"edit":

if \( D\text{msg} \cap \{ (m,l) \} = \emptyset \) then

\[
D\text{msg} := \{ (m,l) \} \cup D\text{msg}
\]

end if

\[
v := \text{SelectVocItem}()
\]

if \( v = \emptyset \) then

\[
do \text{while } z := \emptyset
\]

\[
\text{read}(v)
\]

\[
z := \text{AddVocItem}(v)
\]

end do

end if

\[
S\text{msg}(m,l) := v
\]

\[
x := m
\]

"delete":

\[
D\text{msg} := \neg \{ (m,l) \} \cap D\text{msg}
\]

\[
x := m
\]

end case

end do

3.2 Domains

3.2.1 Select domain

This function enables the user to select an element \( d \) from Domains set \( D \). If a selection is made, the function returns the element \( d \); otherwise, it returns the empty set.

\[
\text{SelectDomain}()
\]

\[
x := \emptyset
\]

\[
\tilde{D} := L
\]

do \( \forall d \in \tilde{D} \)

\[
\text{display}(d, D\text{type}(d), D\text{lng}(d))
\]

\[
\text{read}(\text{choice})
\]

if \( \text{choice} = "\text{select}" \) then

\[
x := \{d\}
\]

end if
Deletion fails if \( d \) is assigned to a data element.

The set \( D \). If deletion succeeds, the function returns the modified \( D \); otherwise, it returns the empty set. Deletion fails if \( d \) is assigned to a data element.

\[ \text{DelDomain}(d) \]
\[ x := \emptyset \]
end if
end do
return(x)

### 3.2.2 Delete domain

This function deletes an element \( d \) from Domains set \( D \). If deletion succeeds, the function returns the modified \( D \); otherwise, it returns the empty set. Deletion fails if \( d \) is assigned to a data element.

\[ \text{DelDomain}(d) \]
\[ x := \emptyset \]
if \( \{d\} \cap D = \{d\} \) then
if \( \{t\} \text{Tdom}(t) = d\} \cup \{s\} \text{Sdom}(s) = d\} = \emptyset \) then
\[ D := \neg\{d\} \cap D \]
\[ D_{\text{Dtype}} := \neg\{d\} \cap D_{\text{Dtype}} \]
\[ D_{\text{Dlng}} := \neg\{d\} \cap D_{\text{Dlng}} \]
x := D
else
display(“domain \( d \) can not be deleted”)
endif
else
display(“domain \( d \) not found”)
end if
return(x)

### 3.3 Entities

#### 3.3.1 Add entity

This function inserts a new element \( e \) in Entities set \( E \) and assigns a unique message \( m \) to it. If addition succeeds, the function returns the modified \( E \); otherwise, it returns the empty set.

\[ \text{AddEntity}(e) \]
\[ x := \emptyset \]
if \( \{e\} \cap E = \emptyset \) then
E := \{e\} \cup E
m := \text{SelectMessage}()
if \( m = \emptyset \) or \( m \in D_{\text{Enam}}^{-1} \) then
z := \emptyset
do while \( z = \emptyset \)
read(m)
z := AddMessage(m)
end do
endif
D_{\text{Enam}} := E
Enam(e) := m
x := E
else
display(“entity \( e \) exists”)
endif
return(x)

#### 3.3.2 Delete entity

This function deletes an element \( e \) from Entities set \( E \). If deletion succeeds, the function returns the modified \( E \); otherwise, it returns the empty set. Deletion fails if there are tags of \( e \) that participate in authority links.

\[ \text{DelEntity}(e) \]
x := \emptyset
if \( \{e\} \cap E = \{e\} \) then
T := \{y\} \text{Tent}(y) = e\}
if \( \{D_{\text{Auth}} \cap T\} \cup \{t\} \text{Taust}(t) = 0, t \in T\} = \emptyset \) then
do while \( t \in T \)
\[ \text{DelTag}(t) \]
end do
E := \neg\{e\} \cap E
D_{\text{Enam}} := E
x := E
else
display(“entity \( e \) can not be deleted”)
endif
else
display(“entity \( e \) not found”)
end if
return(x)

### 3.4 Tags and Subtags

#### 3.4.1 Add tag

This function inserts a new tag \( t \) in Tags set \( T \). Tag \( t \) is assigned to an entity \( e \) and is given a distinct name within the tags of \( e \). A domain is also selected, but authority status, occurrence status and repetition status are simply initialized. If the insertion succeeds, the function returns the modified \( T \); otherwise, it returns the empty set.

\[ \text{AddTag}(t, e) \]
x := \emptyset
if \( \{t\} \cap T = \emptyset \) then
if \( \{e\} \cap E = \{e\} \) then
m := \text{SelectMessage}()
M := \{m\} \text{Tnam}(y) = m, \text{Tent}(y) = e\}
if \( m = \emptyset \) or \( m \in M \) then
do while \( z = \emptyset \)
read(m)
z := AddMessage(m)
end do
delete
T := \{t\} \cup T
D_{\text{Tent}} := T
\text{Tent}(t) := e
D_{\text{Tnam}} := \{t\} \cup D_{\text{Tnam}}
\text{Tnam}(t) := m
d := \text{SelectDomain}()
D_{\text{Tdom}} := \{t\} \cup D_{\text{Tdom}}
\text{Tdom}(t) := d
D_{\text{Tocc}} := \{t\} \cup D_{\text{Tocc}}
\text{Tocc}(t) := 0
D_{\text{Trep}} := \{t\} \cup D_{\text{Trep}}
\text{Trep}(t) := 0
D_{\text{Taust}} := \{t\} \cup D_{\text{Taust}}
\text{Taust}(t) := 0
x := T
else
display(“entity \( e \) not found”)
end if
3.4.2 Delete subTag

This function deletes an element s from Subtags set S. If deletion succeeds, the function returns the modified S; otherwise it returns the empty set. If s is the only subtag of the parent tag t, t is reformed to a simple tag.

\[ DelSubTag(s) \]

\[ x := \emptyset \]

if \( \{s\} \cap S = \emptyset \) then

if \( s \cup \{z\} Stag(z) = Stag(s) \) = s then

\[ D_{\text{dom}} := Stag(s) \cup D_{\text{dom}} \]

\[ d := \text{SelectDomain()} \]

\[ T_{\text{dom}}(Stag(s)) := d \]

\[ D_{\text{occ}} := Stag(s) \cup D_{\text{occ}} \]

\[ T_{\text{occ}}(Stag(s)) := 0 \]

end if

end if

else

display(“subtag s not found”)

end if

return(x)

3.4.3 Update tag

This function updates tag’s properties. It allows the modification of a tag’s message or a tag’s domain and the conversion of occurrence, repetition and authority properties to a different status. It also enables the manipulation of subtags and authority links. If an update is effected, function returns the updated entity t; otherwise, it returns the empty set. The algorithm presented in this section, it is a part of the UpdateTag algorithm that illustrates the manipulation of authority links.

\[ UpdateTag(t) \]

\[ x := \emptyset \]

read(choice)

case choice of

“change authority status”:

if \( \{q\} T_{\text{auth}}(q) = \{t\} \) = \( \emptyset \) read(aust)

do while \( \text{aust} \notin \{0, 1, 2\} \)

read(aust)

end do

\[ T_{\text{auth}}(t) := \text{aust} \]

else

display(“authority status can not be changed”) end if

“create authority link”:

if \( \{t\} \cap D_{\text{auth}} = \emptyset \) then

\[ t_s := \text{SelectTag()} \]

do while \( t_s = \emptyset \) or \( T_{\text{auth}}(t_s) \neq t \) or \( T_{\text{dom}}(t_s) \neq T_{\text{dom}}(t) \)

\[ t_s := \text{SelectTag()} \]

end do

\[ D_{\text{auth}} := \{t\} \cup D_{\text{auth}} \]

\[ T_{\text{auth}}(t) := \emptyset \]

else

display(“an authority link already exists”)

end if

“delete authority link”:

if \( \{t\} \cap D_{\text{auth}} = \{t\} \) then

\[ D_{\text{auth}} := \emptyset \]

\[ T_{\text{auth}}(t) := \emptyset \]

else

display(“no authority link”)

end if

return(x)

4 A PRACTICAL EXAMPLE

To illustrate the functionality of the proposed data model, we present an example to demonstrate a database that manages book and publisher data.

The entity set \( E \) contains elements \( e_{\text{book}} \) and \( e_{\text{publ}} \) (Fig. 1a). In this example, a book is described by ISBN, title, author, publisher and year of publication. A publisher is described by a publisher ID, corporate name, address and phone. The address is a complex tag (Fig. 1b).

![Figure 1: a. Function Tent  b. Function Stag](image-url)

A distinct message is assigned to every entity, tag and subtag using \( E_{\text{msg}}, T_{\text{num}} \) and \( S_{\text{msg}} \) function (Fig. 2). Different tags (subtags) can share the same message as long as they do not reside in the same entity (tag).

Every message describes a conceptual object that can be translated into any of the supported languages by the \( S_{\text{msg}} \) function. In our example, the supported
Figure 2: Functions Enam, Tnam and Snam

languages are English and French (Fig. 3).

Figure 3: Function Smag

Simple tags and subtags hold data which is derived from specific domains. Matching is performed by the functions Tdom and Sdom (Fig. 4). The complex tag t1addr has no domain assigned to it because its subtags are based on specified domains.

Figure 4: Functions Tdom and Sdom

Every simple tag and every subtag has an occurrence status, which is mandatory (value 1) or optional (value 0). Assignment is performed by the functions Tocc and Socc. The complex tag t1addr has no occurrence status as it has no domain (Fig. 5a). On the other hand, repetition status is required for all tags and subtags. Value 1 is given to repeating tags. In our example, the repeating tags are t1phon and t1auth (Fig. 5b).

Figure 5: a. Functions Tocc, Socc b. Functions Trep, Srep

Authority links between tags rely on their domain values and their authority status. Linked tags must be defined on the same domain. Moreover, authority links have to result to a tag with value 1 for authority status. Value 2 is assigned to tags with contents which can be revealed. This assignment is performed by Taust action (Fig. 6a). Function Auth is responsible for creating authority links. In our example, there is an authority link between t1publ and t1phid (Fig. 6b).

Figure 6: a. Function Taust b. Function Auth

5 CONCLUSIONS AND FURTHER RESEARCH

The design of data models that evolve with time is still a major problem today. While user requirements change quite frequently, databases continue to show little flexibility in supporting these changes in their structures and data organization. Research carried out in the direction of supporting schema evolution and schema versioning has proved inefficient in the long-term. In this paper we proposed an integrated set-theoretic model for database systems that forms a framework for defining a structure (unified schema) that eliminates completely the need for reorganization at the logical level. We presented its structure, its features and we demonstrated some of its operations with algorithms that can be applied at the logical level (metadata). The next step is to investigate efficient data storage, data retrieval and manipulation operations, as well as data integrity rules and performance issues.
6 REFERENCES


