CUDL Language Semantics: Authority links

Nikitas N. Karanikolas[1], Maria Nitsiou[2] and Emmanuel J. Yannakoudakis[2]

[1] Technological Educational Institution of Athens, Athens, Greece
{nnk}@teiath.gr
[2] Athens University of Economics and Business, Athens, Greece
{mbnit,eyan}@aueb.gr

Abstract. Conceptual Universal Database Language (CUDL) is a new language designed to manage dynamic database environments, which conform to the Frame DataBase model (FDB). FDB is a generic database model (oversubscribe both the Entity-Attribute-Value and the Nested Relational). CUDL is not only an FDB database language but it is mainly an agent that provides an abstraction level superior to the logical level. CUDL permits the users to conceive a database schema where single fields (tags) accept repetitions (list of values), entertain subfields and also permit to entertain an entire table in the place of a single field. In this paper we explain the advanced power of the FDB - CUDL that emanates from particular improvements to specific parts of the generic FDB model. More precisely, we investigate ameliorations to the "authority links" set of FDB and explain the outgrowth of these ameliorations, to define easily unambiguous and robust relationships between database entities. We also present the novel ability of the model to manipulate data normalizations and data explanations.

1 Introduction

In previous work [27], [28] there has been an investigation of dynamically evolving database environments and corresponding schemata [2], [3], [7], [20], [21], [23], [25], [26], allowing storage and manipulation of variable number of fields per record, variable length of fields, data subfields, multiple value fields, etc. The ultimate goal of Yannakoudakis et al [27], [28] was to make the design and maintenance of a database a much simpler task for database designers, so as that they would not have to put in a lot of effort to design the database and later they would not have to pay extra special attention and work on database changes. This has resulted in a new framework for the definition of a universal database schema that eliminates completely the need for reorganisation at both logical and internal levels, even when the slightest modification in the database requirements must occur. This new framework was called FDB [27].

The management however and operation of this model (framework) is laborious and time-consuming as the user would have to put in a lot of strain to understand and be familiar with the use of the proposed model, meaning the structures and organisation of it (metadata and data), as well as the processes of the management of elements that compose it. For this reason we focused our research in finding an efficient and easy way for the communication of users with the model [19], [22], [9]. This has resulted in creating a language which can bridge the gap between the user and the FDB model, in other words can help the user to manipulate the applications that have been created based on the proposed model [29], [30], [15]. This language was called CUDL [29]. By the use of CUDL, which encapsulates methods with data structures, an FDB management system can execute complex meta-data and data manipulation operations to retrieve and transform information. FDB developers can write complete database applications with the modest amount of effort [29], [30], [15].

In [15] we introduced the syntax and semantics of the CUDL language. There we focused mainly in presenting and analysing the statement of value retrieval (in the schema and the data). In [18] we focused mainly in presenting and analysing the syntax and semantics of the CUDL statements used for value modification (in the schema and the data).

1.1 Motivation

The declaration of relationships between data structures is one of the most significant tools for the enforcement of data validation procedures in any database model. The relational model (through its language SQL) has introduced methods (constraint statements or sub-statements) that permit the declaration of relationships. The following is an example that contains a relationship declaration statement [4]:

```
CREATE TABLE Books (
   BookID INT NOT NULL PRIMARY KEY,
   AuthorID INT NOT NULL,
   BookName VARCHAR(100) NOT NULL,
   Price FLOAT NOT NULL
)

CREATE TABLE Authors (
   AuthorID INT NOT NULL PRIMARY KEY,
   Name VARCHAR(100) NOT NULL
)

ALTER TABLE Books
ADD CONSTRAINT fk_author
FOREIGN KEY (AuthorID)
REFERENCES Authors (AuthorID)
```

The importance of relationships is of so great value that software development leaders have provided relationship enhancements and also statements (or sub-statements) for the declaration of relationship enhancements. The following, modified, sub-statement is a case of such enhancement (provided for MS SQL Server):

```
REFERENCES Authors (AuthorID) ON DELETE CASCADE
```

Obviously the need for relationship declarations exists also for the FDB model and the CUDL language. This need becomes more significant for the FDB-CUDL model because the relationships between entities, in most cases, are implemented without the introduction of new tables (sets in the FDB-CUDL terminology). Without having methods to declare relationships, the user would face a refuting stage where the model is self-explained (the user can consult only tag_attributes and subfield_attributes and carry off the data model) but the data relationships are totally undocumented.

To cope with this need, the FDB model introduced the Authority_links set. However, in its primary version the FDB model [27] had some restrictions. Some improvement was suggested recently [15] but this improvement was far from a complete solution. In our effort for a complete solution, for the documentation of data relationships, a conjecture aroused. According to this conjecture, we could use the relationship declarations structures to also declare authority controls that reduce the variability of expressions used for the same instance of an identity.

2 Background

2.1 FDB

FDB is a generic database schema, with a specific unique form, where we use only one database schema for every application. Whatever the entities needed for the application, no new tables are introduced. We merely define virtual tables (sets) corresponding to the needed entities having virtual fields (tags) and subfields. The FDB model schema has the form shown in Table 1 (note that primary keys are underlined).

The terminology used in FDB is the following:

Entity: a collection of related objects that have certain attributes
Tag: an object used to represent some attribute of an entity
Subfield: an object associated to a specific tag, used to represent a sub-attribute of the tag that belongs to a certain entity
Compo tag: a composite tag hosting subfields
Simple tag: a tag that does not host subfields
Frame object: A specific instance of an entity, meaning an instance holding data that describe a certain case (view of the world) that belongs to the entity. It uniquely identifies this case inside the entity. (We could say that it is analogous to the tuple in the relational model).

We will display a simple example that concerns the application of a Video Club where only two entities (Videos and Customer) are needed. The Videos entity is implemented with an FDB set having seven attributes (tags). The tags

Table 1. The FDB model schema

Languages	(language_id, lang_name)
Datatypes	(datatype_id, datatype_name)
Messages	(message_id, language, message)
Entities	(frame_entity_id, title)
Tag_attributes	(entity, tag, title, occurrence, repetition, authority, language, datatype, length)
Subfield_attributes	(entity, tag, subfield, title, occurrence, repetition, language, datatype, length)
Authority_links	(From_entity, Auth_tag1, To_entity, Auth_tag2, mn_data_loc)
Catalogue	(Entity, Frame_object_number, Frame_object_label Temp_stamp)
Tag_data	(Entity, Frame_object, Tag, Repetition, Chunk, tdata)
Subfield_data	(Entity, Frame_object, Tag, Tag_repetition, Subfield, Subfld_Repetition, Chunk, sdata)

DVD_code, Year, Language and Description are simple tags with single values (without repetitions). It is evident from the names of tags what their contents are. The tags Title and Category are also simple tags but permit multiple values (repetitions). In other words, these tags operate as lists of values. A single instance (frame) of Videos can have more than one title and more than one category that belongs to. The last tag is the Actors tag which is a compo tag with two subfields (Actor1 and Actor2). The Actors tag has a single value (no repetitions) and also its subfields accept only a single value. The Customer entity is implemented with an FDB set having five tags. The tag Cust_code is a simple tag with single values. The tag DVD_code is also a simple tag but permits multiple values (repetitions). This tag is used to hold the codes of the Videos instances that a specific customer (an instance of Customer) has rent. The tags Name and Telephone are compo tags and each of them has two subfields (First name and Last name, Mob phone and Home phone, respectively). Both tags permit a single value (no repetitions) and also their subfields accept only a single value. The last tag is the Address tag which is a compo tag with five subfields (Street, No, Region, City and p.c.). The Address tag accepts repetitions (operate as a list of addresses) but its subfields accept only single values (for example, each address has a single city that belongs to). The FDB application schemata are self-explained, because all the above information is declared in four real FDB tables, the entities, the tag_attributes, the subfield_attributes and the Messages sets. The information, mentioned above, for the Video Club application, is defined in the previously stated four real FDB tables. The contents of the later three tables, for the Video Club application, are shown in Tables 2a, 2b and 2c.

All the data of the Video Club application are stored in barely two real FDB schema tables (namely tag_data and subfield_data). We need no new tables to store each entity's individual data.

Table 2a. Content of the Tag_attributes for Video Club application

Entity	Tag	Title	Occur-rence	Repetition	Authority	Language	Datatype	Length
1	1	2	M	R	N	1	1	40
1	2	3	M	N	N	1	2	NULL
1	3	4	M	N	N	1	5	NULL
1	4	7	M	N	Y	1	1	6
1	5	8	M	R	N	1	1	15
1	6	9	M	N	N	1	1	300
1	7	10	M	N	N	1	1	10
2	1	24	M	N	N	1	1	8
2	2	12	M	N	N	1	5	NULL
2	3	15	M	N	N	1	5	NULL
2	4	18	M	R	N	1	5	NULL
2	5	7	M	R	Y	1	1	6

Table 2b. Content of the Subfield_attributes for Video Club application

Entity	Tag	subfield	Title	Occur-rence	Repetition	Language	Datatype	Length
1	3	1	5	M	N	1	1	40
1	3	2	6	M	N	1	1	40
2	2	1	13	M	N	1	1	20
2	2	2	14	M	N	1	1	20
2	3	1	16	M	N	1	2	NULL
2	3	2	17	M	N	1	2	NULL
2	4	1	19	M	N	1	1	10
2	4	2	20	M	N	1	2	NULL
2	4	3	21	M	N	1	1	12
2	4	4	22	M	N	1	1	10
2	4	5	23	M	N	1	2	NULL

2.2 CUDL

CUDL was designed to manage dynamic databases (schema evolution databases) and also Generic database schemata. The language provides the users a higher abstraction level than the logical abstraction level and can be exploited by simple Generic database schemata (like the Entity Attribute Value - EAV) and more sophisticated ones (like the Frame DataBase - FDB). However, it is mainly designed to exploit all the structures of the FDB model with convenience and effectiveness. Without CUDL, the management and operation of Generic database schemata (like FDB) would require from the user (administrator, developer) a very good acquaintance of the proposed model, the structures and organisation of it as well as the processes of the management of elements that compose it. Otherwise, it would be very difficult and sometimes impossible to carry out even simple operations like the simplest retrieval of information. With the abstrac-

Table 2c. Content of the Messages for Video Club application

Message_id	Language	Message
1	1	Videos
2	1	Title
3	1	Year
4	1	Actors
5	1	Actor1
6	1	Actor2
7	1	DVD_code
8	1	Category
9	1	Description
10	1	Language
11	1	Customer
12	1	Name
13	1	First name
14	1	Last name
15	1	Telephone
16	1	Mob phone
17	1	Home phone
18	1	Address
19	1	Street
20	1	No
21	1	Region
22	1	City
23	1	p.c.
24	1	cust_code

tion level that CUDL introduces, users keep away from difficult programming tasks that the Generic database schemata impose. For example, there is a need for many joins (self-joins) or many queries (or views) and then intersections of the results, in order to retrieve and project data. The CUDL abstraction level removes any such difficulty and let the users conceive the data with more flexible structures than the simple fields of the relational model. The users conceive the data attributes as lists of values, composite values with subfields, etc. Together with this higher perception of data CUDL language preserves the schema evolution characteristic of Generic database schemata. Now we shall portray one frame for each of the entities Videos and Customers of the Video Club application, the way the user apprehends it (CUDL abstraction level). The frames shown in Tables 3a and 3b correspond to instances of the entities defined in Tables 2a, 2b and 2c.

In Tables 4a and 4b we illustrate a more complex example, referring to the Medical informatics domain.

In previous works [15], [17], [18] we have described the syntax and semantics for CUDL data definition and data retrieval statements. However a single data retrieval statement could be helpful for the reader to understand how the CUDL abstraction level is materialized through statements. The following CUDL data retrieval statement is used in order to search for doctors with code like ‘I%8’ treating patients with code like ‘A?15’ (in our medical example):

# Find data when entity = ‘Incident’ and tag = ‘Patient code’ restr data like ‘A?15’ and tag = ‘Incident doctors’ restr data like ‘I%8’

Table 5 portrays the output of this statement.

Table 3a. A Videos frame object

Title	Movie1	
	A Movie	
Year	2003	
DVD_code	Vid01	
Category	Police movies	
	Horror movies	
Description	A description for Movie1...	
	Description for Movie1 continued	
Language	English	
Actors	**Actor1**	**Actor2**
	Al Patsino	Robert DeNiro

Table 3b. A Customer frame object

Cust_code	Cust0002				
DVD_code	Vid02				
	Vid03				
Name	**First name**	**Last name**			
	Nikitas	Salamastras			
Telephone	**Mob phone**	**Home phone**			
	6975106132	2106512345			
Address	**Street**	**No**	**Region**	**City**	**p.c.**
	Papagoy	43	Ilioupoli	Athens	16715
	Ag. Spydidonos	1	Aegaleo	Athens	12210

3 Liaisons

3.1 Relationships

The simplest form of M:N relationships exists in the example of Customers and Videos (Section 2.1). There, each customer can rent (during the whole period of his/her membership) a number of videos and of course each video can be rented a number of times. This M:N relationship can be easily implemented as a list of

Table 4a. An Incident frame object

Incident_code	S001
Date started	13/5/2007
Date ended	20/5/2007
Patient code	A001
Social institute code	T001
Incident doctors	I001 I002 I079

Incident operations

Op code	**Op time started**	**Op time ended**	**Op date**	**Op doctors**
E002	13:35	15:05	14/5/2007	I001 I005 I100 I065
E015	12:00	13:00	16/5/2007	I012 I100 I032

Laboratorial Examinations

LE code	**LE date**	**LE time**	**LE result**
UREA	15/5/2007	10:00	32,4 mg/dl
UREA	15/5/2007	14:30	32,5 mg/dl
UREA	16/5/2007	08:00	31,6 mg/dl
CREA	15/5/2007	10:00	1,17 mg/dl
CREA	16/5/2007	08:00	1,08 mg/dl
PROT	15/5/2007	10:00	7,19 g/dl
PROT	16/5/2007	08:00	6,95 g/dl

Radiological Examinations

RE code	**RE date**	**RE time**	**File Path**
U/S Kidney	16/5/2007	12:00	\\FS1\RIS\Uaz34.tif

values hosted by any of the related entities and without the need of an individual set (table in the relational terminology). However the list of values should be hosted exclusively in one of the related entities and not both. Otherwise, we will face redundancy problems. To be more specific, we have two solutions. The first solution is to use a tag, named for example DVD_Code, in the Customer entity and permit this field to have more than one value (repetitions). This tag will operate as a list with all the codes of the videos (DVDs) that a particular customer (represented by the particular frame) has rent. This is also the solution

Table 4b. Two Doctors frame objects

doctor code	I001
name	Manos
surname	Grigoropoulos

doctor code	I002
name	Theodoros
surname	Pachopoulos

adopted and presented in the example of section 2.1. The second solution is to use a tag, named for example Cust_code, in the Videos entity and permit this field to have more than one value (repetitions). This tag will operate as a list with all the codes of the customers that have rented a particular video (represented by the particular frame). Whatever is the adopted M:N solution, it is materiazed by relating two tags coming from the related entities (one tag from the first and another tag from the second entity). Obviously, there is a need to document someway our decision (adoption). This can be happen with an entry in the Authority_links set, which could be something like the triple (Videos.DVD_code, Customer.DVD_code, R) or (Videos.Cust_code, Customer.Cust_code, L). The first of them defines that there exists a M:N relation between the tag DVD_Code of the Videos entity and the tag DVD_code of the entity Customer whereas the data implementing the relationship are kept in the later of them. The second triple defines that there exist a M:N relation between the tag Cust_code of the Videos entity and the tag Cust_code of the Customer entity whereas the data implementing the relationship are kept in the former of them.

A second form of M:N relationships exists in our Electronic Patient Record (EPR) example. For the kind of relationship we are interested to display only the sets Doctors and Incidents should be examined. Let us assume that the number of frames existing in the mentioned sets are 6 and 5, respectively. Let us also

Table 5. Results of a CUDL data retrieval statement

Entity	**Frame_object**	**Tag**	**Repetition**	**Data**
Incident	1	Patient code	1	A115
		Incident doctors	1	I018
			2	I128
			3	I008
	12	Patient code	1	A015
		Incident doctors	1	I038

assume that the five frames of the set Incidents have 3, 4, 1, 2 and 2 repetitions in the "Incident operations" tag, respectively. The Incidents set hosts the tag "Incident operations" (the operations that a patient undertook during the period of an incident) which is a tag that hosts subfields and can have repetitions (meaning none, one or more operations took place). The subfields of the "Incident operations" are "Op code", "Op time started", "Op time ended", "Op date" and "Op doctors". The relationship we are interested is between the "Incident operations" and the Doctors. There, in each operation of an incident, a number of doctors are participating and of course each doctor could have participated in many operations, either in the same incident or in different incidents. The difference from the previously examined relationship (between Customers and Videos) is that here we have a M:N relationship between a tag (not a set as previously) and a set. To emphasize this distinctive relationship we will examine the possible values of it. In a common M:N relationship (like the one of Customers with Videos) the maximum number of possible values is the product of the number of frames of the first set by the number of frames of the second set (otherwise M x N). Contrariwise here, the maximum number of possible values is Sum(M_i) x N, where M_i is the number of repetitions (values) of the related tag in the i^{th} frame (of the set hosting the tag). According to our assumptions, the number of possible values in this relationship is Sum(3, 4, 1, 2, 2) x 6 = 12 x 6 = 72. We will use the symbol S(M):N to denote the present (second) form of relationship. Here, there exists also the need to document someway the relationship between "Incident operations" and Doctors. This can happen with an entry in the Authority_links set. The next triple (Incident."Incident operations"."Op doctors", Doctors."doctor code", L) is such an Authority_links entry. It defines that there exists a S(M):N relation between the subfield "Op doctors" of the tag "Incident operations" of the entity Incident and the tag "doctor code" of the entity Doctors and the data implementing the relationship are kept in the former of them.

In the last (second) case of M:N relationships we have a S(M):N relationship where the data implementing the relationship are kept in the S(M) side. There is an apparent question. Is it possible to have an S(M):N relationship where the data implementing the relationship are kept in the N side? The answer is yes and in order to make it clear we will provide an alternative design of the EPR example. In other words we will modify the design of our EPR example. We will replace the subfield "Op doctors" with the without repetitions subfield "Incident operation ID". The later subfield will have unique values for the whole Incident set, in contrast of having unique values only for the range of a whole frame. We will introduce another tag, named "Operations participated", for the entity "Doctors". This tag will permit repetitions and will hold the unique values of the subfield "Incident operation ID" where the doctor participated. In order to document someway this (third) M:N relationship we will add an entry in the Authority_links set having the form (Incident."Incident operations"."Incident operation ID", Doctors."Operations participated", R). This entry defines that there exists a S(M):N relation between the subfield "Incident operation ID" of the tag “Incident operations” of the entity Incident and the tag “Operations participated” of the entity Doctors and the data implementing the relationship are kept in the later of them (the N side).

It is obvious that someone will ask about a fourth case of M:N relationships, do there exist cases that need to be handled with Sum(M_i):Sum(N_i) relations? The answer is yes. We will put forward the Projects-Employees example that verifies our utterance. Tables 6a and 6b exhibit two representative frames of the entities Projects and Employees.

Table 6a. A Projects frame object

Project_code	Proj077
Title	Zeus
Budget	317,000
Actions	

Actor	**Action**	**Deadline**
E00103	Software analysis	17/10/2007
E00402	Software requirements	22/01/2008
E00702 E00903	Program code	23/04/2008

Table 6b. An Employees frame object.

Emp_code	E007
Family_name	Giorgos
Last_name	Georgiou
Speciality	

Spec_code	**Spec_Descr**
E00701	Delphi Programming
E00702	Java Programming
E00703	Rational Rose Usage

It is obvious from table 6 that the tag Speciality (of the Employees set) is a compo tag that contains the alternative specialities of an employee. It is composed by two subfields (named Spec_code and Spec_descr, respectively) and its first subfield (Spec_code) has unique values for the whole Employees set. The subfield Actor (of the tag Actions of the set Projects) permits repetitions and obtains values originating from the subfield Spec_code (of the tag Speciality of the set Employees). In this example potentially, each repetition of the Actions tag of each Projects frame can be combined with each repetition of the Speciality tag of each Employees frame. In other words the maximum number of feasible combinations is the product of the sum of the number of actions of all projects (Sum(M_i)) by the sum of the number of specialities of all employees (Sum(N_i)). The data implementing the relationship are kept in the Actor subfield (the Sum(M_i) side). In order to document someway this (fourth) M:N relationship we will add an entry in the Authority_links set having the form (Projects.Actions.Actor, Employees.Speciality.Spec_code, L). As in the previous cases of relationships a simpler denotation for the current one is S(M):S(N).

3.2 Authority controls

The authority control is a concept coming from a long time ago [6], before computers appeared. The idea of an authority control is to reduce the variability of expressions used to characterize the same identity. Authority control implies the verification and standardization of access points in a paper or electronic file [8], [24]. For example the standardization of author names in a library information system is a well-known application of authority control. In general an authority control is implemented with a structure named authority file or authority list. The authority file is a collection of authority records, where each authority record is used to define a single name, title or entity (from this point forward we will use the term entity to refer to a single name, title or entity). Each authority record is composed by four components. These are: the Heading or the Authorized form, the See references, the See Also references and the Justification. Heading refers to the form of an entity that has been chosen as the used form of entering data in an index or catalogue. See references are the other (alternative) forms of the entity that might appear somewhere. See references are the forms of the entity that have been deprecated in favor of the Heading. See Also references, are pointers to other forms of the entity that are authorized. See Also references are most commonly used to point to earlier or later forms of an entity. For example See Also can contain the Heading of some journal before the journal renamed. The Justification is a set of statements for documenting the sources of information used to determine both the authorized and the deprecated forms of the entity. Some of the justification statements cite the title, publication date and the source. Some others determine the location where the entity exists, the year of birth of the entity, etc.

It is obvious that an authority file can be easily implemented as an FDB-CUDL set with four tags: Heading (simple text tag, without repetitions), “See references” (text tag with repetitions), “See Also references” (text tag with repetitions) and Justification (text tag with repetitions). Table 7 is an example frame of an authority file implemented as an FDB-CUDL set (named ‘Author Authority file’).

Having defined an authority file, all is left to do is notify FDB-CUDL with an authority control relationship. Particularly, which entity - tag (or subfield)

Table 7. An Authority file frame object

Heading	Chomsky, N.
See references	Avram Noam Chomsky
	Noam Chomsky
	Chomsky, Noam
	Chomsky, N.
See Also references	*empty*
Justification	Born December 7, 1928
	http://en.wikipedia.org/wiki/Noam_Chomsky
	http://www.chomsky.info

combination is under whose authority control. This can happen with an entry of the form (Entity.Tag, AuthorityName, A) or an entry of the form (Entity.Tag.Subfield, AuthorityName, A) in the authority_links set. After the last notification, the CUDL language will interfere transparently and will undertake every step needed to replace any deprecated form (provided by the user) with the corresponding Heading.

3.3 Combining Relationships and Authority controls

It is obvious that the form of Authority_links presented in table 1, is not adequate to record all the forms of M:N relationships and also record the authority control relationships. The following, modified, form of Authority_links is able to hold any one of the previously discussed relationships:

Authority_links (from_entity, from_tag, from_subfield, to_entity, to_tag, to_subfield, relationship_type)

It remains to specify the range of values of the relationship_type attribute. Table 8 contains the selected range of values and the necessary explanations. It is obvious that a relationship of type 1 can be reversed to (redeclared as) a relationship of type 2. The same is true for the couples of relationships 3, 6; 4, 5; 11, 12 and 14, 15. The relationships with types 11, 12, 14 and 15 are special cases of other relationships. This is reflected by their type value (11 is a special case of 1, 12 is a special case of 2, and so on). The special cases are needed when we characterize an instance of the relationship with extra attributes. An example of such case arises if we replace the simple tag (with repetitions) DVD_code of the entity Customer (see table 3b) with a composite tag, named Rents (see table 9), having three subfields. The meaning of the subfield DVD_code is obvious and the other two subfields ("Start date" and "End date") add extra information to each instance of the relationship (rental period). In this case we have to modify

Table 8. The range of values for the attribute relationship_type.

type	**Explanation**	**Symbol**
1	Simple M:N, data implementing the relationship are kept in the M side	M:N/L
2	Simple M:N, data implementing the relationship are kept in the N side	M:N/R
3	S(M):N, data implementing the relationship are kept in the S(M) side	S(M):N/L
4	S(M):N, data implementing the relationship are kept in the N side	S(M):N/R
5	N:S(M), data implementing the relationship are kept in the N side	N:S(M)/L
6	N:S(M), data implementing the relationship are kept in the S(M) side	N:S(M)/R
7	S(M):S(N), data implementing the relationship are kept in the S(M) side	S(M):S(N)/L
8	S(M):S(N), data implementing the relationship are kept in the S(N) side	S(M):S(N)/R
11	Simple M:N, data implementing the relationship are kept in subfield in the M side	M:N/L↓
12	Simple M:N, data implementing the relationship are kept in subfield in the N side	M:N/R↓
14	S(M):N, data implementing the relationship are kept in subfield in the N side	S(M):N/R↓
15	N:S(M), data implementing the relationship are kept in subfield in the N side	N:S(M)/L↓
20	Authority control over Tag	A/T
21	Authority control over Subfield	A/S

the declared relationship (Videos, DVD_code, null, Customer, DVD_code, null, 2) with (Videos, DVD_code, null, Customer, Rents, DVD_code, 12).

Table 9. The composite tag Rents.

Rents	**DVD_code**	**Start date**	**End date**
	Vid02	15/1/2008	17/1/2008
	Vid03	20/2/2008	23/2/2008

Some extensions of the CUDL language, in order to be able to declare relationships, are needed. The following are examples of the 'Add relationship' statements of CUDL:

# Add relationship type = '2' from_entity = 'Videos' from_tag = 'DVD_code' to_entity = 'Customer' to_tag = 'DVD_code'
# Add relationship type = '12' from_entity = 'Videos' from_tag = 'DVD_code'

to_entity = ‘Customer’ to_tag = ‘Rents’ to_subfield = ‘DVD_code’
# Add relationship type = ‘3’ from_entity = ‘Incident’ from_tag = ‘Incident operations’ from_subfield = ‘Op doctors’ to_entity = ‘Doctors’ to_tag = ‘doctor code’
# Add relationship type = ‘20’ from_entity = ‘Books’ from_tag = ‘Author’ to_entity = ‘Author Authority File’

4 Discussion - Conclussions

In this paper we investigate the Authority_links set of the FDB model, having a double objective. The first aim is to define easily unambiguous and robust M:N relationships between database entities, as relationships between database entities is one of the most important tools for the enforcement of data validation procedures in any database model. The second aim is to declare authority controls that reduce the variability of expressions used for entities. In other words, to provide verified and standardized access points of entities and to materialize the consistent use and maintenance of information.

In the case of the FDB-CUDL model the relationships between entities, in most cases, are implemented without the introduction of new tables. The M:N relationships can be easily implemented as a list of values (hosted exclusively by one of the related entities to avoid redundancy) without the need of a new separate set.

An authority file can be easily implemented as an FDB-CUDL set with four tags. Moreover, a notification to FDB-CUDL is required indicating which entity - tag (or subfield) combination is under whose authority control. This notification enforces the transparent intervention of the semantic CUDL mechanisms which undertake any step needed to replace any deprecated form (provided by the user) with the corresponding authorized form.

All the information concerning relationships and authority controls is located in one single set, the Authority_links set. This way the user can see all the links between fields and subfields existing in the data source listed all together and she/he does not have to search the entire database to find these links. The design of data source relationships therefore becomes much simpler (in the CUDL abstraction level) in contrast with the relational or the object oriented database model. Information retrieval becomes faster because we look in only one set. The access plans and other implementation and operation details for relationships and authority controls are handled in a standard way, by the CUDL language interpreter, and are not reflected in the logical model. This contrasts with common practice for SQL DBMSs in which performance tuning often requires changes to the logical model.

References

1. Amini, M.R., Gallinari, P.: Automatic Text Summarization Using Unsupervised and Semi-supervised Learning. In: PKDD’2001, 5th European Conference on Prin-

ciples of Data Mining and Knowledge Discovery. (2001)

2. Andany, J., Leonard, M., Palisser, C.: Management of schema evolution in database. In: Proc. of the 17th VLDB Conf., Barcelona (1991) 161–170
3. Bratsberg, S.E.: Unified Class Evolution by Object-Oriented Views. In: proceedings of the Internaltional Conference / the Entity-Relationship Approach, Springer Verlag. (1992) 423–439
4. Chigrik, A.: Using Microsoft SQL Server Constraints. Available online at: http://www.mssqlcity.com/Articles/General/using_constraints.htm.
5. Chuang, W.T., Yang, J.: Extracting Sentence Segments for Text Summarization: A Machine Learning Approach. In: SIGIR'2000, 23th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. (2000)
6. Cutter, C.A.: Rules for a Dictionary Catalog. Washington, D.C.: U.S. Government Printing Office (1904)
7. Ferrandina, F., Meyer, T., Zicari, R., Ferran, G., Madec, J.: Schema and database evolution in the O2 object database system. In: Proceedings of the 21th International Conference on Very Large Databases (VLDB '95), Zurich (1995) 170–181
8. Garshol, L.M.: Metadata? Thesauri? Taxonomies? Topic maps! Making sense of it all. Journal of Information Science **30**(4) (2004) 378–391 Available online at: http://www.ontopia.net/topicmaps/materials/tm-vs-thesauri.html.
9. Hennessy, M.: The semantics of programming languages: an elementary introduction using structural operational semantics. John Wiley & Sons, New York (1990)
10. Karanikolas, N.N., Mantzaris, S.L.: Innovative directions in information retrieval. In: HERMIS'92, Hellenic Research on Mathematics and Informatics, Athens (1992)
11. Karanikolas, N.N., Skourlas, C.: Computer Assisted Information Resources Navigation. Medical Informatics and the Internet in Medicine **25**(2) (2000)
12. Karanikolas, N.N., Skourlas, C.: Shifting from legacy systems to a Data Mart and Computer Assisted Information Resources Navigation framework. In: 5th International Conference On Enterprise Information Systems (ICEIS), Angers, France (April 23-26, 2003)
13. Karanikolas, N.N., Skourlas, C., Christopoulou, A., Alevizos, T.: Medical Text Classification based on Text Retrieval techniques. In: Proceedings of the 1^{ST} International Conference on Medical Informatics and Engineering - 1^{ST} MEDINF, Craiova Medicala **5** (supplement 3), Craiova, Romania (2003) 375–378
14. Karanikolas, N.N., Skourlas, C.: Naive Rule Induction for Text Classification based on Key-Phrases. In: Data Mining VI. Data Mining, Text Mining and their Business Applications. Volume 35 of WIT Transactions on Information and Communication Technologies, Skiathos, Greece, WIT Press (2005) 175–181
15. Karanikolas, N.N., Nitsiou, M., Yannakoudakis, E.J., Skourlas, C.: CUDL language semantics, liven up the FDB data model. In: Eleventh East-European Conference on Advances in Databases and Information Systems (ADBIS 2007), Varna, Bulgaria (September 29 - October 03, 2007)
16. Karanikolas, N.N.: Low cost, cross-language and cross-platform Information Retrieval and Documentation tools. Computing and Information Technology (CIT) Journal **15**(1) (2007)
17. Karanikolas, N.N., Nitsiou, M., Yannakoudakis, E.J., Skourlas, C.: Conceptual Universal Database Language (CUDL) and Enterprise Medical Information Systems. In: 10th International Conference on Enterprise Information Systems - ICEIS 2008, Barcelona, Spain (12 - 16, June 2008)
18. Karanikolas, N.N., Nitsiou, M., Yannakoudakis, E.J., Skourlas, C.: CUDL language semantics: updating FDB data. Submitted for publication (2008)

19. Subieta, K.: Semantics of Query Languages for Network Databases. ACM Transactions on Database Systems (TODS) **10**(3) (1985) 347 – 394
20. McKenzie, E., Snodgrass, R.: Scheme evolution and the relational algebra. Information Systems **15**(2) (1990) 207–232
21. Mohamed, A.N., Estublier, J.: Schema evolution in software engineering databases - a new approach in Adele environment. Computers and Artificial Intelligence **19** (2000)
22. Niemi, T., Christensen, M., Jarvelin, K.: Query language approach based on the deductive object-oriented database paradigm. Journal of Information & Software Technology **42**(11) (2000) 777–792
23. Roddick, J.F.: Schema evolution in database systems: an annotated bibliography. SIGMOD Record **21**(4) (1992) 35 – 40
24. Savoy, J.: Bibliographic database access using free-text and controlled vocabulary: an evaluation. Information Processing & Management **41** (2004) 873–890
25. Zdonik, S.B.: Object-oriented type evolution. In Bancilhon, F., Buneman, P., eds.: Advances in Database Programming Languages, ACM Press / Addison-Wesley (1990)
26. Zicari, R.: A Framework for Schema Updates In an Object-Oriented Database System. In: Proceedings of the Seventh International Conference on Data Engineering. (1991) 2 – 13
27. Yannakoudakis, E.J., Tsionos, C.X., Kapetis, C.A.: A new framework for dynamically evolving database environments. Journal of Documentation **55**(2) (1999) 144–158
28. Yannakoudakis, E.J., Diamantis, I.K.: Further improvements of the framework for dynamic evolving of database environments. In: Proceedings of the HERCMA 2001 5th Hellenic - European Conference on Computer Mathematics and its Applications, Athens, Greece (2001)
29. Yannakoudakis, E.J., Nitsiou, M.: A new conceptual universal database language (CUDL). In: 2nd International Conference From Scientific Computing to Computational Engineering (2nd IC-SCCE), Athens, Greece (2006)
30. Yannakoudakis, E.J., Nitsiou, M., Skourlas, C., Karanikolas, N.N.: Tarski algebraic operations on the frame database model (FDB). In: Proceedings of the 11th Panhellenic Conference in Informatics (PCI 2007), Patras, Greece (2007)