CUDL Language Semantics, Liven Up the FDB Data Model

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

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

Abstract. The semantics of a new language (CUDL – Conceptual Universal Database Language), designed to manage dynamic database environments, are presented. This language conforms to the FDB (Frame DataBase) data model. The FDB model supports conceptual database record processing using a universal schema that eliminates completely the need for reorganization at logical and internal levels when changes in a database application occur. The CUDL language offers a focused, flexible, efficient and highly expressive environment that also takes advantage of the FDB's characteristics of a continuously evolving database. It provides the ability to define and manipulate database information and changes that can be easily navigated, ensuring full maintainability of all applications. We present a formal definition of the CUDL language's semantics with example interpretations. The basic features, query specification and interpretation, object manipulation, query language constructs and query processing techniques used in the language are discussed.

1 Introduction

In previous work [20], [21] there has been an investigation of dynamically evolving database environments and corresponding schemata, allowing storage and manipulation of variable length data, a variable number of fields per record, variable length records, manipulation of authority records and links between records and fields, and dynamically defined objects (relations in the traditional sense). This has resulted in a new framework for the definition of a unified schema that eliminates completely the need for reorganisation at both logical and internal levels. Retrieval of data is optimised through self-contained storage chunks that also vary dynamically. This new framework was called FDB.

The FDB schema is a schema in the relational model in which a subset of its data (that is mainly stored in a subset of tables) functions as a descriptor (driver), that is to say as metadata that determine the structure - way of handling of the rest of the data. This structure of organisation allows easy schema evolution. The management however and operation of this model is laborious and time-consuming and it requires from the user a very good acquaintance of the proposed model, the structures and organisation of it (metadata και data) as well as the processes of the management of elements that compose it. For this reason we wanted to create a language which will help the user to manipulate the applications that have been created based on the proposed model [22] [23].

By encapsulating methods with data structures, an FDB management system can execute complex analytical and data manipulation operations to search and transform

Y. Ioannidis, B. Novikov and B. Rachev (Eds.): Local Proceedings of ADBIS 2007, pp. 1-16

information. FDB developers can write complete database applications with a modest amount of additional effort.

Let us analyse this a bit more. The user conceives the data of an FDB system via CUDL ***as an extension*** of the relational model. What the user conceives gives him tables (entities in the FDB model), rows (frames in the FDB model) and columns (tags in the FDB model). More specifically, it gives him the sense of the organisation of data under the form of tables ***with certain moreover extensions***. Concretely these extensions refer to that each field can have ***multiple values***, meaning that a cell as it is meant in the relational model in the CUDL language can be a ***list of values***. Also a cell in CUDL can have ***multiple subfields***. Moreover it can have a combination of the two precedents, where ***a cell can by itself entertain a table***.

Another conceivable extension that experiences the user from the CUDL language is that the fields of what the user considers as data of the relational model can have ***variable length***. And finally the CUDL language entertains the possibility of dynamic evolution of a database which is an inherited attribute of the underlying FDB system.

As we know, the definition of a programming language consists of at least two parts, the syntax and the semantics. The syntax is concerned with the form of expressions, which are allowed in the language. On the other hand the semantic definition could describe the expression or program or it could describe how to execute or evaluate them [8].

In this paper we introduce the syntax and semantics of the CUDL language. The CUDL query language is developed into a language over the FDB data model.

The formal specification of programming languages is an established field in computer science. Syntax is often formally defined using a BNF grammar that describes the set of legal sentences in the language. Semantics, however, is less often defined with any formal rigour. The definition of semantics has the following objectives:

- to allow for a deep understanding of the language
- if the language is not yet fixed, the formal description leads to the possibility of improvement and generalization of language constructs
- to make implementation much easier. (Indeed, a deep understanding of the semantics is a prerequisite for the implementation of any language)
- to make possible the formal study of language properties [11]

The language examined here defines the semantics by a mixture of informal rules, and examples. The data source used as example is intended only to illustrate the main concepts, and does not cover all the language features**.** We think that its essential features can be introduced by giving expressions for our querying primitives [14]. Its functionality is illustrated by means of numerous useful CUDL queries. The novelty of CUDL lies in its ability to smoothly combine schema and data querying while exploiting all FDB modelling features.

Section 2 contains a brief review of the FDB model. It also presents related works. In section 3 we exploit some aspects of the FDB model that required improvement and the improvements that we introduce. Section 4 describes the CUDL language syntax and semantics. Section 5 contains discussion, conclusion and future work.

2 FDB and Related works

In [20], [21] authors have proposed a new framework for building databases without the need for schema reorganization when changes in the database must occur.
The work that has been carried out aimed at the design and implementation of a framework that can be attached to an existing relational model in such a way as to offer hybrid facilities for both relational and object oriented platforms. The ultimate objective was to establish a new framework that can lead us to an independent database management system with built-in utilities [20], [21].
Such a database schema is indeed difficult for any user to maintain. Therefore we need a tool which can offer easy, focused, flexible management of this schema with the minimum effort from the user
In order to provide such a tool, we focused our research over the past few years, in the design of a new language, which we call CUDL [22], that offers a targeted, focused and easy to use management system for the FDB database framework. We decided to go with the design of a totally new language as from our research it became apparent that, so far, no existing database language could capture all of the FDB model capabilities and distinctiveness. Because of this distinctiveness it was also obvious that no existing database language could offer an easy to use and flexible tool that could focus entirely on the FDB data model without any excess elements.
Schema evolution has attracted a considerable amount of attention in the database community [1], [2], [4], [5], [7].
There are some approaches that use the temporal database theory and incorporate temporal support within the meta-database, thus permitting the modelling of changing schemata [15], [12]. Applications written to access an old schema can still work after the schema changes, because past schemata are retained [15]. The queries are guaranteed to return the same results as if they were executed before the schema change took place. Data inserted in the format of the new schema are not accessible through the old programs, so the functionality provided is limited to old data. However, changes still need to be made to the applications in order to cope with changes in the real world being modelled.
Most of the approaches proposed fall, particularly in the context of object-oriented databases [3], [4], [6], [2], [17], [18], [19], [13], [16]. Theses approaches can be divided into three categories based on the external representation of the structure of the objects in the database (object schema) to application programs and interactive users and the internal representation of the objects in the underlying database.
In the area of multi-value, multi attribute fields in their work, Karanikolas and Scourlas [9], present a document management, text classification and text retrieval system (named CAIRN) that combines the ideas of multi-value (multi-row) fields, multi attribute (multi-column) fields and the combination of both of them. In other words it also permits to use two dimensional tables in the position of a single field.
In his work, Karanikolas [10] presents also a Field Based QBE where the user can take advantage of the multi-value / multi attribute fields and express queries requiring to retrieve records (frames in the FDB terminology) that contain two (or more) distinctive values concurrently in the same record (frame).

3 FDB Enhancements and Clarifications

In this section we propose an improved FDB schema and discuss the improvements in detail. The basic schema of the proposed framework has the following form (note that primary keys are underlined):

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

3.1 Handling Relations Between Data

It carries into effect 1:M and M:N relations without the need for intermediate sets (tables). 1:M relations are carried into effect very easy with tags that accept repetitions. M:N relations are also carried into effect with repetitions. In these cases there is the capability for the data of the M:N relation to be stored in either one of the two entities. In order to avoid the danger of redundancy (by the creation of a tag with repetitions in both entities) we added the attribute mn_data_loc (with values in the domain {L|R}) in the authority_links set which defines the entity where the tag with repetitions is stored.

3.2 Multilinguality of the Model and Datatyping

Aiming at the easy transfer of an FDB-CUDL application in another language we moved on certain improvements - adaptations of the FDB of model. In the new improved version of the FDB model we have replaced the fields tag_indicator2 and tag_indicator3 in the set tag_attributes with language and datatype respectively. Also we have added the field language and have replaced the field subfield_indicator with datatype in the subfield_attributes set.
The second (datatype) indicator is concerned with the type of data having the data stored in the tag_data and the subfield_data set.

3.3 Relating Tags and Subfields

In the previous presentations of the FDB model it had not become quite clear how the data that exist in the set tag_data are connected with their corresponding data that are likely to exist in the set subfield_data. This is made possible via the values that exist under the attributes entity+frame_object that exist in the two sets. That is to say, that each data that corresponds in the combination entity+frame_object+tag+subfield in set subfield_data is the data that corresponds in the combination entity+frame_object+tag in set tag_data.

3.4 Composite Datatype

A restriction that we place for the FDB model and materialise with language CUDL, as we report below, is that when a combination of entity+tag has subfields in the datatype attribute we should declare as datatype the value “composite”. In different case (that is to say when we have declared another datatype) we should not be allowed to add any subfield_attributes for this particular entity+tag. In the previous version of the FDB model the user had the right to determine even in the tag_attributes that contained subfield_attributes a specific type of data. But we considered that this did not reflect suitably the reality of an application since subfield_attributes that belonged in a tag_attribute could be of a different type of data either from each other or with regard to the tag_attribute they belonged.

3.5 Check Repetitions

A second restriction that we have added for the FDB model is that when it has not been given in set tag_attributes and subfield_attributes under the field repetition the value ‘R’ it should not be allowed to insert repeated data in the sets tag_data and subfield_data.

3.6 Numerical datatypes

One more restriction that we have placed for the FDB model is that when we have declared as types of data in the set tag_attributes and subfield_attributes certain types (such as real, integer etc) it should not be allowed to declare a value for this datatype’s corresponding length attribute.

3.7 Other improvements

Other improvements that we made in respect with the previous version of the FDB model [20], [21] are listed below:

- The set indicator2_data is renamed in languages, because we considered that the new name represents better the content of this set.
- The set indicator3_data it was renamed in datatypes, because, also, we considered that the new name represents better the content of this set.
- The Set indicator1_data has been removed, because we considered that the information that this set contained was not essential for our model. Also, we have

removed the attribute tag_indicator1 from the tag_attributes set that it referred in the values of the set indicator1_data.

4 The CUDL Language Syntax and Semantics

4.1 General description

CUDL allows the specification of queries in a high-level, declarative manner. For example, to select rows from a data source, the user needs only to specify the criteria that he wants to search by; the details of performing the search operation efficiently is left up to the system, and is invisible to the user. The user provides the CUDL statement. Then an appropriate algorithm is applied to perform the task defined by the user statement: This algorithm performs a search in the metadata in order to confirm whether the structural elements (eg. tag_attributes, subfield_attributes) that the user specifies are valid. Afterwards it locates the subset of data that the user determines and finally it executes the statement of the user in the subset of data that he has focused. This structure allows the user/programmer to be less familiar with the technical details of the data and how they are stored, and relatively more familiar with the information contained in the data. This blurs the line between user and programmer, appealing to individuals who fall more into the 'business' or 'research' area and less in the 'information technology' area. The original vision for CUDL is to allow non-technical users to write their own data source queries. The features of CUDL presented here have been partially implemented [22], [23].

4.2 Assumptions

The concept of set is fundamental to CUDL. Every nameable attribute or instance must belong to a set. In other words a set is an object that holds instances and can contain zero or more instances. Database sets can be accessed through their name (as defined in the FDB model). All sets are described by attributes which are not common to all the sets. The user can address these attributes by the names originally given to them in the definition of the FDB model.

4.3 Basic Constructs

Arguments, Datatypes, Languages, System messages.

A valid argument of a CUDL statement can be a string expression in quotes, a reserved keyword and numbers in any form. CUDL provides a fixed set of datatypes. We cannot define new datatypes. CUDL forces us to choose from a given set of predefined datatypes when we create or modify an instance. CUDL provides a fixed set of languages. However new languages can easily be defined. All System messages are stored in the set called sys_interface. It is essential that the system messages are loaded prior to any implementation of any data source attempts as they will be used as messages to the user.

Statements.

At the highest level, CUDL language statements can be broadly categorized as follows into three types: Data Definition Language (DDL) which defines the structure of the data, Data Control Language (DCL) which defines the privileges granted to the database and Data Manipulation Language (DML) which retrieves or modifies data for the users.

Data definition.

The most basic statements of DDL are the 'Create' and 'Remove' statements. 'Create' causes an object (a set (or sets), for example) to be created within the data source. 'Remove' causes an existing data source to be deleted, irretrievably, or an existing element (set) within a data source to de deleted (along with the instances in it), also irretrievably. A required condition for the use of the DDL statements is that the sets of metadata have been created which happens once.

Data control.

The second group of CUDL keywords is the Data Control Language (DCL). DCL handles the authorisation aspects of data and permits the user to control who has access to see or manipulate data within the data source. Its two main keywords are: 'Permit' that authorises a user to perform an operation or a set of operations and 'Restrict' that removes or restricts the capability of a user to perform an operation or a group of operations.

Data manipulation.

The third group of keywords is the Data Manipulation Language (DML). This group contains four basic statements: FIND, ALTER, DELETE, ADD. These statements are used most often by application developers. DDL and DCL statements are commonly used by a database designer and database administrator for establishing the database structures used by an application. We should mention that we will show the results of the CUDL statements as these will be presented to the user and not as they are stored in the sets of the FDB application.

- *Find*

The most frequently used operation in data sources is the data retrieval operation which retrieves instances from a set. The FIND statement specifies which attributes to include in the result set. Find is used to retrieve zero or more instances from one or more sets in a data source. In the following, we present the semantics of the find statement for retrieval from schema. A hash symbol (\#) always introduces a statement.

Find entities,
We will have the result shown in Table 1.

Table 1.

Entity
Videos
Customer

This query seeks for values in the sets of FDB. The result is the values that satisfy the search criteria or all the values of a set (as it appears in the example above). The resulting information of the query is drawn in only one virtual collection. The algorithm that our prototype CUDL interpreter performs for the above example is the following:

```
Select title from entities
Insert the results of the previous query into a new virtual set
called set1
If set1 = ø
    Return a suitable message to the user
Else
    For every tuple in set1
        Select message from messages where message_id = title
        Replace the value of title in the set1 with the value of
        message
    Endfor
    Return set1 to the user
Endif
```

Find entities except ‘videos’

Find tag_attributes
It retrieves all the tag_attributes.

Find tag_attributes group to entity

Find tag_attributes when title = ‘DVD_code’
The result of this statement is shown in Table 2.

Table 2.

Entity	Tag	Title	Occurrence	Repetition	Authority	Language	Datatype	Length
Videos	4	DVD_code	M	N	N	English	Char	6
Customer	5	DVD_code	M	R	N	English	Char	6

Find tag_attributes except ‘name’

In the following, we present the semantics of the find statement for the retrieval from Schema in combination with Boolean Operators.

Find tag_attributes when entity = ‘videos’ and title = ‘actors’

Find tag_attributes when entity = ‘customer’ and repetition=‘r’ and datatype=‘char’
The result of this statement is shown in Table 3.

Table 3.

Entity	Tag	Title	Occurrence	Repetition	Authority	Language	Datatype	Length
Customer	5	DVD _code	M	R	N	English	Char	6

In the following, we present the semantics of the find statement for data retrieval.

Find data when entity = ‘customer’ and tag =‘cust_code’
This statement retrieves the codes of the customers (the projected field is cust code). This will result in the set shown in Table 4.

Table 4.

Entity	Frame_object	Tag	Repetion	Chunk	Tdata
customer	1	cust_code	1	1	Cust0001
	2		1	1	Cust0002
	3		1	1	Cust0003
	4		1	1	Cust0004

Informally this statement takes the sequence of all customers and creates a sequence of their ‘cust_code’. The result of typing in this query is that a sequence of ‘cust_CODE’ is printed out. Looking more closely at the query, it is built out of three sets, entities, tag_attributes and tag_data. Customer is a value contained in the entities set and has a tag_attribute ‘cust_code’ that generates a sequence of the codes of customers. Here we have to state that a fourth set, the set messages, is invoked in the implementation algorithm hidden behind the query. The implementation algorithm is the following:

```
SELECT  message_id FROM messages WHERE message = 'customer'
Select frame_entity_id from entities where title = message_id
Insert the results of the previous query into a new virtual set
called set1
SELECT  message_id FROM messages WHERE message = 'cust_code'
Select tag_attributes.tag from tag_attributes where title =
message_id
Insert the results of the previous query into a new virtual set
called set2
If (set1 = ø) or (set2 = ø)
    Return a suitable message to user
Else
```

```
    Select entity, frame_object, tag, repetition, chunk, tdata
    from tag_data where entity = frame_entity_id and
    tag_attributes.tag = tag_data.tag
    Insert the results from the previous SQL statement into a
    new virtual set called tag_data1
    If tag_data1 = ø
        Return a suitable message to user
    Else
        For every tuple in tag_data1
            Replace the entity value with the value 'customer'
            Replace the tag value with the value 'cust_code'
        Endfor
        Return the set tag_data1 to the user
    Endif
Endif
```

Find data when entity = 'videos' and tag = 'title'
This statement retrieves the titles of the videos (the projected field is 'title').

Find data when entity = 'videos' and tag = 'title' frame
Notice that by adding the keyword *frame* we take all the data of all the tags for every frame that meets the user requirements. The condition `and tag='title'` is unnecessary and can be excluded from the statement.

Find data when entity = 'videos' and tag = 'title' restr data like 'M%' and tag = 'year' restr data = '2003'
We have to mention that the conditions can be composed by a where part and a restriction part. For example the condition `and tag = 'title' restr data like 'M%'` has a where part `tag = 'title'` and a restriction part `restr data like 'M%'`. The fields that take place in the where part of the conditions also participate in the results (projected fields).

The algorithm behind this query is:

```
Select message_id FROM messages WHERE message = 'videos'
Select frame_entity_id from entities where title = message_id
Insert the results of the previous query into a new virtual set
called entity_set
Select message_id FROM messages WHERE message = 'title'
Select tag_attributes.tag from tag_attributes where title =
message_id
Insert the results of the previous query into a new virtual set
called tag_set1
{Obviously tag_set1 is either empty (ø) or has one single value}
If tag_set1 ≠ ø
    Set tag1 = tag_set1
endif
Select message_id FROM messages WHERE message = 'year'
select tag_attributes.tag from tag_attributes where title =
message_id
```

```
Insert the results of the previous query into a new virtual set
called tag_set2
{Obviously tag_set2 is either empty (ø) or has one single value}
If tag_set2 ≠ ø
    Set tag2=tag_set2
endif
If (entity_set = ø) or (tag_set1 = ø) or (tag_set2 = ø)
    Return a suitable message to the user
Else
    select entity, frame_object,tag,repetition,chunk, tdata from
    tag_data where entity = frame_entity_id and (tag_data.tag
    =tag1 and tdata like 'M%') and (tag_data.tag = tag2 and
    tdata ='2003')
    Insert the results from the previous SQL statement into a
    new virtual set called tag_data1
    If (tag_data1 = ø)
        Return a suitable message to the user
    Else
        For every tuple in tag_data1
            Replace the entity value with the value 'videos'
            If tag_data1.tag=tag1
                Replace the tag value with the value 'title'
            Endif
            If tag_data1.tag=tag2
                Replace the tag value with the value 'year'
            Endif
        Endfor
        Return the set tag_data1 to the user
    Endif
Endif
```

Find data when entity = 'videos' and tag = 'title' restr data like 'M%' and tag = 'year' rest data = '2003' and tag = 'category'
This statement performs selection of data based on title and year and projection based on title, year and category.

Find data when entity = 'videos' and tag = 'title' restr data like 'M%' and tag = 'year' rest data = '2003' HIDE or tag = 'category' restr data like 'Com%'
Selection based on title and year or based on category, projection based on title and category.

Find data when entity = 'customer' and tag = 'address'
This statement results in the addresses of all the customers.

Find data when entity = 'customer' and tag = 'address' and subfield = 'Last name' restr data = 'Karanikolas'
This statement will result in the address of the customer with Last name = 'Karanikolas'.

Find data when entity = ‘videos’ and tag = ‘category’ restr data LIKE ‘Comed%’ and subfield = ‘actor1’ restr data = ‘Al Patsino’
Here there is a choice based on the tag ‘category’ and the subfield ‘actor1’. Finally, we project the same fields.

Find data when subfield = ‘street’ and subfield = ‘ No’ and subfield = ‘region’ and subfield = ‘p.c.’ restr like ‘1%’
Here we take the subfields ‘street’, ‘No’, ‘region’ and ‘p.c’ when the restriction that has been determined on the subfield ‘p.c’ is verified. We observe that the entity is not determined. In this case the system finds which set has all the determined subfields. If more than one entities exist that allocate all the four subfields the question is very general and it is not executed. In the opposite case results are returned. From here we can see that there is the need the names of the subfields in same entity to be unique. Otherwise the syntax of the find statement would have to become more complex so that the user will be required to determine the name of the tag as well as the name of the subfield in the where part condition.

- *Add.*

The add statement serves two purposes: the first is to import values in sets where the metadata are kept, while the second is to import values in sets where the data are kept. For the first purpose the user must provide the values for all the attributes in the record in the order they are defined. The user can, however, optionally set only values for a subset of attributes, and then default values are inserted in the rest of the attributes.
The first example is an example of the addition of a new tag in the tag_attributes set (a set where metadata are kept). Let us for example assume that there is no tag ‘language’ for the entity ‘videos’. The statement:

Add tag_attributes entity = ‘videos’ title = ‘language’
will result in the tuple shown in Table 5 stored in the tag attributes set.

Table 5.

Entity	Tag	Title	Occurrence	Repetition	Authority	Language	Datatype	Length
1	7	9	O	R	N	1	1	100

Note that for the attributes *Occurrence, Repetition, Authority, language, datatype, length* default values are being added automatically. However if we provide the statement:

Add tag_attributes entity = ‘videos’ title = ‘language’ language = ‘greek’ length = 10
We will get the result shown in Table 6 stored in the tag attributes set.

Table 6.

Entity	Tag	Title	Occurrence	Repetition	Authority	Language	Datatype	Length
1	7	9	M	N	N	2	1	10

add subfield_attributes entity = ‘customer’ tag = ‘telephone’ title = ‘mob phone’

In the following we are giving examples for the second purpose of Add. The next example shows the addition of data for a whole frame in the entity ‘videos’.

Add tag_data entity = ‘videos’ tag = ‘title’ tdata = ‘Movie1’ new frame
Add tag_data entity = ‘videos’ tag = ‘title’ tdata = ‘A Movie’ when tdata = ‘Movie1’
…
Add subfield_data entity = ‘videos’ tag = ‘actors’ subfield = ‘actor1’ sdata = ‘Al Patsino’ when tdata = ‘Movie1’
Add subfield_data entity = ‘videos’ tag = ‘actors’ subfield = ‘actor2’ sdata = ‘Robert DeNiro’ when tdata = ‘Movie1’

Note that by adding the key-phrase ‘new frame’ the data are inserted in a new frame. Otherwise if those keywords are omitted the data are inserted in an already existing frame that the user must specify by adding the keywords *when tdata = ‘<some value>’*. If the user does not use the specification referred above then a suitable message is presented to him and the statement is not executed.

- *Delete.*

Delete removes a specified instance or a group of instances from a set. Here we have an example of a delete statement for the removal of an instance from a metadata set:

Delete tag_attributes when entity = ‘videos’ or title = ‘actors’
This will result in the deletion from the set tag_attributes of all the values that correspond to the combination of the entity ‘videos’ and the tag ‘actors’. The deletion will only occur under the condition that there do not exist data for the ‘actors’ tag and also the specified tag does not participate in any authority link (set authority_links).

Below we have an example for the deletion of a whole frame from the entity ‘videos’ (remove an instance from a data set):

Delete tag_data when entity = ‘videos’ and tag = ‘DVD_code’ restr tdata = ‘Vid01’ whole frame

We have to mention that the set ‘catalogue’ is affected as well, by this delete statement. As the set catalogue is used to keep the values of the entity and each frame for this entity that the set tag_data contains, since a whole frame from the set tag_data was erased the corresponding entity and frame from the set catalogue should be erased too.

- *Alter.*

The alter statement serves two purposes: the first is to alter values in sets where the metadata are kept, while the second is to alter values in sets where the data are kept. Below we present examples for the first purpose of the alter statement (alter values in sets where the metadata are kept)
Let us assume that we want to have the possibility to insert two or more different addresses for one customer. This means that we have to set the repetition attribute in the subfield_attributes set to 'R' for all the subfield_attributes that refer to the address. The CUDL statement we will use is the following:

Alter tag_attributes repetition = 'n' with 'r' when entity = 'customer' and title = 'address'

The next is an example to show the second purpose of alter (alter values in sets where the data are kept).

Alter tag_data set 'title' = 'Movie1' when entity = 'videos' and tag= 'DVD_code' restr data = 'Vid01' and tag = 'title' restr data = 'Movie1'

5. Discussion, Conclusions and Future Work

We have presented a robust language which is capable to present to users the FDB model's features as easily understandable virtual extensions of the relational model. This language has consistent statement constructions that can be easily absorbed. For example the condition part of a find (retrieve data) statement has exactly the same structure with the condition part of an alter (update data) statement. Moreover the CUDL language has great expression power as it is documented in [23].
There are also some other recent efforts for building systems supporting schema evolution, multi-value fields and multi-attribute fields. However our effort has a greater theoretical substructure and can be exploited more widely than other efforts. In order to support our allegation we can mention that, in contrast to CAIRN, CUDL language can be implemented as a driver (or as an extra shell) between an application and a relational DBMS. On the other hand CUDL still has room for improvements and can be benefited from other efforts. For example CAIRN systems permit users to define retrieval conditions that demand the simultaneous existence of two (or more) distinguished values in the same record (frame). This is a feature that CUDL does not support yet. Such features and others, more technical ones, are those that are going to attract our attention in the future. An example of a more technical feature that we are going to attack, is that the parts of a selection restriction (*where part* and *restriction part*) should both become optional.

References

1. Andany, J., Leonard, M., Palisser, C.: Management of schema evolution in database. In Proc. of the 17th VLDB Conf., Barcelona (1991) 161-170

2. Banerjee, J., Kim, W., Kim, H. and Korth H.F.: Semantics and Implementation of Schema: Evolution in Object-Oriented Databases. ACM SIGMOD, Vol. 16, No. 3 (1987) 311 - 322
3. Bertino, E.: A View Mechanism for Object-Oriented Databases. In: Pirotte, A., Delobel, C. and Gottlob G. (eds): Proc. of Advances in Database Technology (EDBT'92) - 3rd international Conference on Extending Database Technology, Lecture Notes in Computer Science, Vol. 580, Springer (1992) 136-151
4. 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
5. Cattel, R.: The object database standard. Odmg 2.0 (Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann, San Francisco, California (1997)
6. Clamen., S. M.: Schema Evolution and Integration. Distributed and Parallel Databases, Special issue on distributed/parallel database object management, Vol. 2, No. 1 (1994) l0l-126
7. Ferrandina, F., Meyer, T., Zicari, R., Ferran, G., Madec, J.: Schema and database evolution in the O2 object database system. Proceedings of the 21th International Conference on Very Large Databases (VLDB '95), Zurich (1995) 170-181
8. Hennessy, M.: The semantics of programming languages: an elementary introduction using structural operational semantics. John Wiley & Sons, New York (1990)
9. Karanikolas, N. and Skourlas, C.: Shifting from legacy systems to a Data Mart and Computer Assisted Information Resources Navigation framework. 5th International Conference On Enterprise Information Systems (ICEIS) Angers, France (April 23-26, 2003)
10. Karanikolas N.: Low cost, cross-language and cross-platform Information Retrieval and Documentation tools. Computing and Information Technology (CIT) Journal, vol. 15, No 1 (2007)
11. Kazimierz Subieta: Semantics of Query Languages for Network Databases. ACM Transactions on Database Systems (TODS) Volume 10, No 3 (1985) 347 – 394
12. McKenzie, E. and Snodgrass, R.: Scheme evolution and the relational algebra. Information Systems, Vol. 15, No. 2 (1990) 207–232
13. Mohamed A.-N., Estublier, J.: Schema evolution in software engineering databases - a new approach in Adele environment., Computers and Artificial Intelligence, Vol. 19 (2000)
14. Niemi, T., Christensen M. and Jarvelin, K.: Query language approach based on the deductive object-oriented database paradigm. Journal of Information & Software Technology, Vol.42, No. 11 (2000) 777-792
15. Roddick, J.F.: Schema evolution in database systems: an annotated bibliography. SIGMOD Record (1992) 35 - 40
16. Monk, S. and Sommerville, I.: Schema Evolution in OODBs Using Class Versioning. SIGMOD RECORD, Vol. 22, No. 3 (1993) 16 - 22
17. Skarra, H. A. and Zdonik, S. B: Type Evolution in an Object-Oriented Database. In Mit Press Series In Computer Systems, Research directions in object-oriented programming, MIT Press, Cambridge, MA, USA (1987) 393 - 416

18. Zdonik, S. B.: Object-Oriented Type Evolution. In Advances in Database Programming Languages, Addison-Wesley (1990)
19. Zicari, R.: A Framework for Schema Updates In an Object-Oriented Database System. Proceedings of the Seventh International Conference on Data Engineering (1991) 2 - 13
20. Yannakoudakis E.J., Tsionos C.X. and Kapetis C.A.: A new framework for dynamically evolving database environments. Journal of Documentation, Vol. 55, No. 2 (1999), 144-158.
21. Yannakoudakis E. J., Diamantis I. K.: Further improvements of the Framework for Dynamic Evolving of Database environments. In Proceeding of the HERCMA 2001 5th Hellenic – European Conference on Computer Mathematics and its Applications, Athens, Greece (2001)
22. Yannakoudakis E. J., and Nitsiou M.: A new conceptual universal database language (CUDL). In 2nd IC-SCCE: 2nd International Conference From Scientific Computing to Computational Engineering, Athens, Greece (2006)
23. 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)