Database Design with Real-World Structures

Nikitas N. Karanikolas[1] and Michael Gr. Vassilakopoulos[2]

[1] Department of Informatics
Technological Educational Institute of Athens, Greece
nnk@teiath.gr

[2] Department of Computer Science and Biomedical Informatics
University of Central Greece, Lamia, Greece
mvasilako@ucg.gr

The simplicity of the relational model types affects Information Systems design. In this paper, we discuss another approach, where the Information System designers would be able to depict the real world in a database model directly. This model provides more powerful and composite data types, as those of the real world. However, more powerful models need query and manipulation languages that can handle the features of the new composite data types. The Conceptual Universal Database Language (CUDL) is such a language that leads to more abstract database design levels. We demonstrate that a database modelled by Entity-Relationship (ER) diagrams can be directly expressed to the CUDL Abstraction Level (CAL), by a set of rules for direct transformation of ER diagrams to CAL. This set consists of 5 rules, proposed in a previous work by us and, in this paper, it is extended with an extra rule that deals with specific situations appearing in practical applications. Consequently, the resulting more powerful and composite data can be directly maintained with the CUDL language. In this way, the development process consisting in conceptual design (ER), transformation to Logical Relational Schema, usage of SQL for data manipulation/retrieval and the reverse steps to present the results in the conceptual level is simplified by conceptual design (ER), transformation to CAL and the usage of CUDL for direct manipulation/retrieval of real world (conceptual) structures.

1. Introduction

A common methodology that is used up to today in the design of relational databases is the specification of a set of attributes in (usually) one and universal relation, the specification of a set of functional dependencies among these attributes and the decomposition of the set of attributes into smaller relations which consist of subsets of the original set of attributes, in order to eliminate update anomalies and reduce data redundancy. During data manipulation, the combination of attributes through the relational join operator is needed.

However, the real world that we are called to model with an Information System (often with the use of a database) seldom incorporates repetitions of data (data redundancy). Thus, we claim that the Information System design should not decompose the real world in its fundamental characteristics, but the Information System designers should be able to portray directly the real world in a model that provides more powerful structures (through composite data types), as those of the real world.

To support composite data types, meta-models (metadata schemata and the corresponding metadata) are needed. Handling of such (composite data) types by the programmer by SQL-like languages and extensions requires knowledge of the internal organisation of both metadata and data, unless a data manipulation language able to manipulate directly the composite data types is employed. The Conceptual Universal Database Language (CUDL) is such a language. We demonstrate that a database modelled by Entity-Relationship (ER) diagrams can be directly expressed to the CUDL Abstraction Level (CAL) (in this paper, we extend the set of rules for direct transformation of ER diagrams to CAL, presented in [8]). Consequently, the resulting more powerful and composite data can be directly maintained with the CUDL language. In this way, the development process consisting in conceptual design (ER), transformation to Logical Relational Schema, usage of SQL for data manipulation/retrieval and the reverse steps to present the results in the conceptual level is simplified by conceptual design (ER), transformation to CAL and the usage of CUDL for direct manipulation/retrieval of real world (conceptual) structures.

The organization of the paper is as follows. Section 2 provides a review of data models that were proposed to overcome weaknesses of the relational model. However, having more powerful database models, but still using data manipulation languages designed only for fundamental data attributes, is a waste of time and resources. Therefore, in Section 3, we present the basic ideas related to CUDL and its abstraction level, CAL [5, 6, 7, 14]. The transformation from CAL to FDB (meta-model, [12, 13]) has been studied in [5, 7]. In Section 4, we present the 5 Rule Set for Transformation from ER to CAL [8] (dealing with the most common cases that appear in real life applications, for the transformation from ER diagrams to CAL), while in Section 5, we propose the addition of an extra rule that is needed for the transformation of a relation between a weak and a strong (not identifying) entity that also has attributes. Section 6 concludes the paper and presents future research possibilities.

2. Data Modeling Approaches

The Generic Data Modeling [1, 4] approach is an outcome mainly emanating from research in the Medical Informatics domain. The fact that, in case of Health Care data maintenance, the amount of information and the complexity of information lead to a huge (Daedal / mazy) conceptual schema, concerned the Medical Informatics scientists. The disadvantage of Generic Data Modeling is that querying the resulting generic logical schema with standard SQL requires multiple statements and considerable intellectual effort, especially when the queries are intended to retrieve data for feeding data analysis tasks (e.g. feeding data mining applications).

The Entity Attribute Value (EAV) data modeling [9, 10] is also an outcome from research in the Medical Informatics domain. The motivation for the research that revealed the EAV data modeling was that, in the medical domain, the number of parameters (facts) that potentially apply to any clinical study is vastly more than the parameters that actually apply to an individual clinical study. Another reason, that motivated the research that revealed the EAV data modeling, was that clinical studies are subject to evolution as a result of medical research. As a consequence, the number of clinical parameters related to a clinical study is always differentiated (and, in most cases, are increased). Thus, the data model should be able to host new clinical parameters for any clinical study, without the need for data (structure) reorganization. Although, EAV data modeling support facts evolution (equivalent to the addition columns in a relational table without the need for any reorganization), it does not support table (entity) evolution.

The fact that the Relational model has difficulties of modeling the real world motivated the development of the Nested Relational data model. Researchers proposed a relational model where Non First Normal Form (NF2) relations are allowed [11]. This extension encompasses the classical 1st Normal Form (1NF) model and adds, to the relational algebra, two basic operators (namely "nest" and "unnest"). Based on the "nest" operator, this proposal allows sets (as the result of "one-attribute" nest operation) and sets of sets (as the result of "multi-attribute" nest operation) as attribute values. NF2 sets are equivalent to simple FDB (explained in the next paragraph) fields with repetitions and NF2 sets of sets are equivalent to composite FDB fields with repetitions. The researchers have also proposed a query language extension for NF2 table definition and manipulation. However, the NF2 presents a number weak points, presented in [8].

Yannakoudakis et al. [12, 13] investigated dynamically evolving database environments and corresponding schemata, allowing storage and manipulation of a variable number of fields per record, variable length of fields, composite fields (fields having subfields), multiple values per field (either atomic or composite), etc. The ultimate goal of their research was to make the design and maintenance of a database a simpler task for database designers, so as that they will not have to put in a lot of effort to design the database and later they will not have to pay special attention and work for database changes. Their research proposed a new framework for the definition of a universal logical database schema that eliminates completely the need for reorganisation at both logical and internal levels, even when major modifications of the database requirements have occurred. This new framework was called Frame Database Model (FDB) [12].

This Universal logical database schema is based on well and strictly defined set of Metadata and data tables and it does not permit any mixture with conventionally stored data. All the entities that are available to the user, along with their attributes, are documented exclusively in the metadata tables and the facts concerning the instances of the entities are recorded exclusively in the data tables of the FDB Universal schema. Another noteworthy feature of FDB is that it supports Schema evolution, both for facts and entities. Moreover the FDB model allocates ways of imprinting strictly connected (hardly related) information with innate (inherent/native) mechanisms, in contrast to the relational model that compels the creation of artifact structures (tables) to represent strictly connected data. The most important fact in the FDB model is that it organizes information without any repetition of values. In order to be more precise, not only it does not proceed in repetitions but it ensures that these cannot be created. Thus the FDB model provides as an inherent feature the no redundancy property.

The weakness of the relational model to manage complex, highly interrelated information motivated the research for Object Databases (ODB) and Object Relational Databases (ORDBs). Both models are described in several textbooks, e.g. [2]. Our personal opinion is that the data management professionals do not like to bother themselves with the required programming overhead needed for classes' construction, inheritance, etc, and consequently they do not decide easily to use an ODB. The other direction, ORBDs, aim to provide solutions for complex and highly interrelated information management, without imposing complicated programming constructions. The SQL3 standard provides an extension to the previous SQL standards, for handling the most of the characteristics added with the ORDBs.

3. The Conceptual Universal Database Language

It is obvious from the plethora of models (presented in the previous section) that there is a need for a DBMS able to provide more powerful data types, as those of the real world complex data. In the first four discussed models, handling of such types by the programmer by SQL-like languages and extensions requires knowledge of the internal organization of both metadata and data [8]. Thus, the ultimate goal should be to provide a data manipulation language able to manipulate directly the composite data types (without bothering the programmer with internal organization details), while permitting the direct expression of restrictions over subfields.

In the approach that we have adopted, the FDB [12, 13] is the underlying model for implementing our goal for a data manipulation language able to manipulate directly composite data types. We preferred the FDB model [6], since it is more compact and well defined than the other models and also supports schema evolution. Karanikolas et al. [5, 7], introduced the syntax and semantics of the CUDL, a database language (both DDL and DML for the FDB model) that supports composite data types, i.e. attributes (tags in CUDL terminology) that combine more than one other sub-attributes (subfields in CUDL terminology). It also supports the existence of multiple values (repetitions) for both tags and subfields. The key difference of CUDL to other approaches is that not only tags, but also subfields and repetitions can be addressed in the CUDL statements. That means that the users can express retrieval restrictions over a specific subfield or can express the modification of a specific subfield in a specific repetition of some tag, etc. In [5], the authors focused mainly in presenting and analyzing the statement of value retrieval (in the schema and the data). More recently [7], they focused mainly in presenting and analyzing the CUDL statement of value modifications in the schema (schema changes) and the data. They also have undertaken the need for relationship declarations [6]. 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. 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. They also use the Authority_links set to declare authority controls and reduce variability of expressions used for the same instance of an identity. All of these (relationship declarations and authority control declarations) are provided through CUDL statements.

Apart from being a data manipulation language able to manipulate directly composite data types, while permitting the direct expression of restrictions over subfields, CUDL allows the application programmer/designer to model the structures of its application with composite data types that are closer to the ER diagrams and sometimes without any decomposition of the ER entities into simpler ones (an example is depicted in [8]).

On the other hand the logical database level of any CUDL based application is the underlying FDB model. Thus, instead of transforming from ER to simple relational tables to provide a logical model for manipulation through SQL, we are able to transform from ER to CUDL Abstraction Level (CAL) entities (namely, CUDL data sets with composite data types). In other words, the classic database design triplet (ER, logical and physical design) is replaced by the quadruple: ER, CAL, logical and physical design, with a fixed logical design (the FDB model).

CAL is the data model that the user conceives. It is a model that supports collections of uniform (congenerous) instances (frame objects in CAL terminology). The structure of each instance (of a concrete collection) is a set of attributes (tags in CAL terminology). In contrast to the relational model, attributes (tags) can have either simple (predefined) data types or can be composed by a set of sub-attributes (subfields in CAL terminology). CAL model also supports the existence of multiple values (repetitions) for both tags and subfields. Each CAL collection of uniform (congenerous) instances is called entity (also called "abstract entity" for reasons explained in [7]). The CUDL language is a database language for defining (DDL) and handling (DML) CAL entities.

Given the above characteristics of CUDL and FDB, it is explained why somebody would prefer to use a CUDL-DB versus an ORDB. The main advantages of CUDL-DB are:

- schema evolution,
- not unused fields (zero repetitions make unneeded the use of null values),
- several real world constructions are neither so complex as to demand data blades (and other extended capabilities of ORDBs), nor natively supported by the relational model (e.g. more than one telephone numbers of a customer require two tables with an 1:M relationship); these "intermediate" (neither simple, nor complex) data constructions do not bother the programmer and are natively supported in CUDL and CAL.

It is obvious, from the above explanation, that the transformation from ER entity types to CAL entities is a direct mapping. However, the transformation from ER relationship types to CAL structures is a more complicated process and we are going to consider it in the next section.

4. The 5 Rule Set for Transformation from ER to CAL

4.1 A Transformation Example

In order to make more evident the need for designing in upper levels we will provide an example of a really complicated ER (corresponding to a real situation) and its transformation to CAL entities (CUDL data sets with composite data types). We are going to present an Electronic Patient Record (EPR) that could be maintained in a real Hospital Information System (HIS). A description of our HIS appears in [8].

The ER diagram of Figure 1 is detailed conceptual depiction of the HIS – EPR database. According to the ER diagram of Figure 1, there is a binary relationship between the Incident and the "Laboratorial Examination" entity types with cardinality ratio M:N. A similar binary relationship between the Incident and the "Radiological Examination" entity types, with cardinality ratio M:N, is also depicted. A third binary relationship with cardinality ratio M:N exists for the Incident and the Doctors entity types. This relationship is responsible for the servant physicians of the incident. The unique ternary relationship of the ER diagram is an identifying relationship of the weak entity type “Incident Operation”. There are two owner entity types that identify the weak entity type “Incident Operation”. These are the Incident and the Operation entity types. There is a binary relationship, with cardinality ratio M:N, between the weak entity type “Incident Operation” and the (strong) entity type Doctors. The later relationship expresses the set of doctors (surgeons, anaesthesiologists, etc) that participate in each “Incident Operation”.

Next, we will provide the CAL entities that the ER diagram of Figure 1 is transformed. The CAL entities are (for composite and multivalued attributes, () and {} are used, respectively, Subsection 3.3.1 of [2]):

<u>Incident</u>
Incident_code, Date_started, Date_ended, Patient_code, SSI_code,
{Lab_examinations (LE_code, LE_datetime, LE_result)},
{Rad_examinations (RE_code, RE_datetime, RE_FilePath)},
{Incident_operations (Op_code, IO_datetime_started, IO_datetime_ended,
{IO_doctors})}, {Incident_doctors}, ICD9_code.

<u>Doctor</u>
Doctor_code, name, surname.

<u>Operation</u>
Op_code, name, cost.

<u>Rad_Examination</u>
RE_code, description, type, cost.

<u>Lab_Examination</u>
LE_code, name, normal_values, cost.

<u>Patient</u>
Patient_code, name, surname, father_name, tax_registration_number,
date_of_birth.

<u>Social_Security_Institute</u>
SSI_code, name, immediate_insured_rate, intermediate_insured_rate.

<u>Diagnosis</u>
ICD9_code, ICD9_description.

4.2 The Initial Transformation Rule Set

The rules for transforming from (relationships types of) ER diagrams to CAL entities (at least for the cases that appear mostly in real applications, like those needed for transforming from the ER of Figure 1 to the CAL entity Incident) that were presented in [8] are the following:

Rule1. Binary relationships with cardinality ratio M:N can be hosted in one of the two related entities as a tag (field) with repetitions. In case that the relationship has no attributes, a simple (not composite) tag with repetitions is capable to store the primary key values of the related (hosted) entity. Whenever the relationship has attributes, a composite tag (composed of the primary key values of the hosted entity and the relationship’s attributes) with repetitions should be used. Obviously, the key of the hosting participant is not needed. This rule applies for the relationship between the Incident and the “Laboratorial Examination” entity types and for the relationship between the Incident and the Doctor entity types. Since we have selected to host the relationships in the Incident entity type, the former relation is implemented with the following composite tag with repetitions:

{Lab_examinations (LE_code, LE_datetime, LE_result)}.

For the same reason, the later relation is implemented with the following simple tag with repetitions:

{Incident_doctors}.

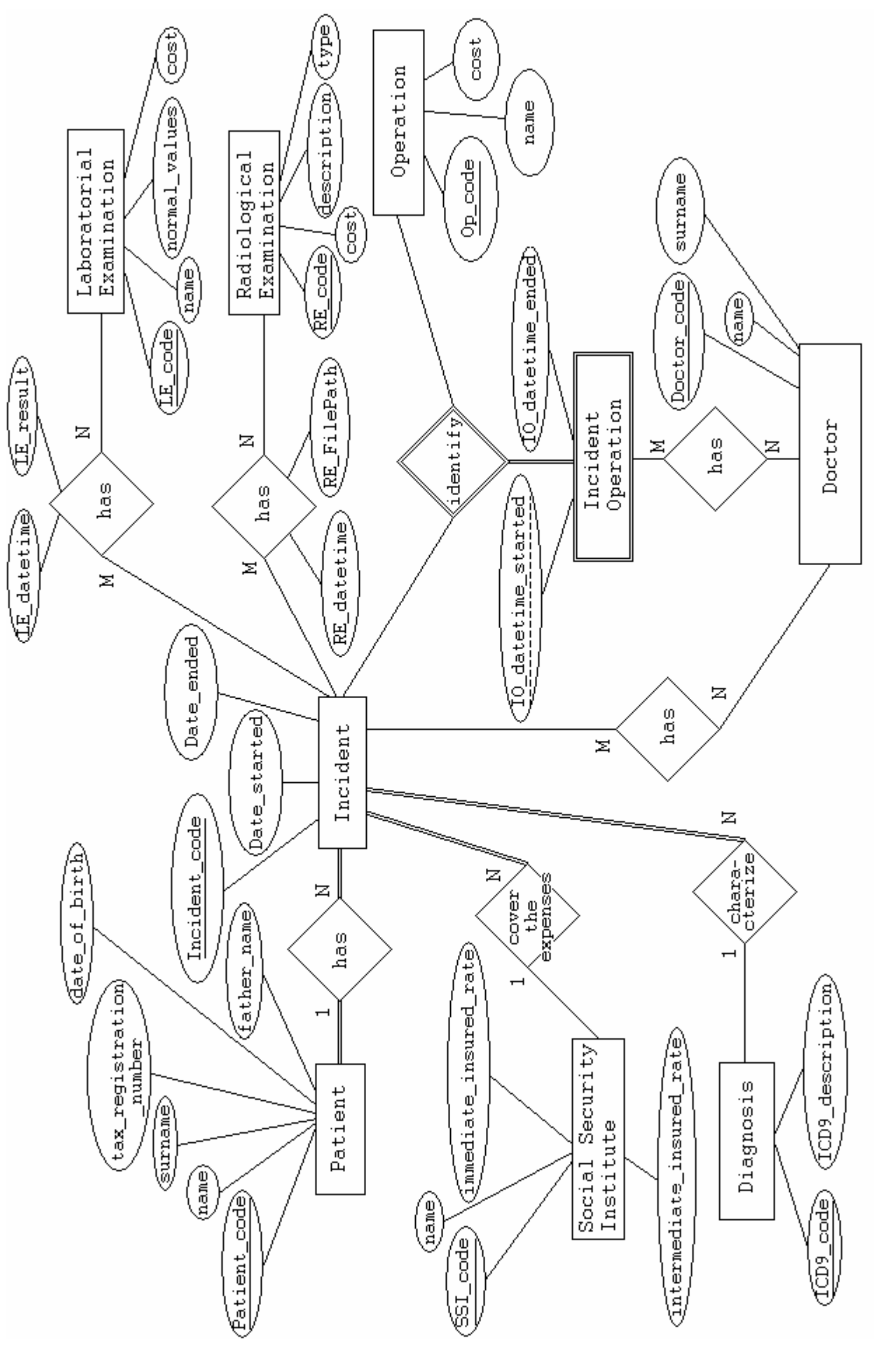

Figure 1: The ER diagram of the HIS-EPR.

Rule 2. Binary identifying relationships can be transformed to a composite tag with repetitions, hosted in the owner entity type. In this case, the composite tag should be composed by the partial key (of the weak entity) and the rest attributes of the relationship. Obviously, the key of the hosting owner entity is not needed. There is no application of this rule in the studied ER diagram.

Rule 3. The previous rule can be extended for ternary identifying relationships. Ternary identifying relationships can be transformed to a composite tag with repetitions, hosted in one of the (two) owner entity types. In this case, the composite tag should be composed by the partial key (of the weak entity type), the key of the hosted owner entity type and the rest attributes of the relationship. Obviously, the key of the hosting owner entity type is not needed. This rule applies for the weak entity type “Incident Operation”. Since we have selected to host the relationships in the Incident owner entity type, the relation is implemented with the Incident_operations composite tag with repetitions:

{Incident_operations (Op_code, IO_datetime_started, IO_datetime_ended, {IO_doctors})}

The partial key of the weak entity type participates in the Incident_operations as the IO_datetime_started subfield. The key of the owner entity type Operation participates in the Incident_operations as the Op_code subfield. The subfield IO_datetime_ended is an attribute of the relationship. The role of the IO_doctors subfield will be explained with the next rule.

Rule 4. Binary relationships with cardinality ratio M:N, relating a weak entity type with some other (strong) entity type, without having relationship attributes, can be transformed to an extra subfield with repetitions of the composite tag implementing the weak entity. This rule applies for the relationship between the weak entity type “Incident Operations” and the entity type Doctor. This rule explains the last constituent of the Incident_operations tag, presented above. (The domain of {IO_doctors} is the power set of the Doctor_code’s domain.)

Rule 5. Binary relationships with cardinality ratio 1:N can be hosted in the N-side entity type, as a tag (field) without repetitions. In case that the relationship has no attributes, a simple (not composite) tag is enough for storing the primary key values of the opposite-side entity type. Otherwise, whenever the relationship has attributes, a composite tag should be used. The primary key of the opposite-side entity type and the relationship’s attributes composes this composite tag. Obviously, the key of the hosting (N-side) participant is not needed. This rule applies for the relationships of the Incident with the entity types Patient, Social_Security_Institute and Diagnosis, respectively. The tags Patient_code, SSI_code and ICD9_code of the CAL entity Incident implement these relationships.

Actually, the rules presented above are also responsible for updating the foreign-key constraints.

5. The New Transformation Rule

In the following we will provide an extension of above set of rules. We are going to provide a rule that applies in a specific subcase of relation between entity types, because this subcase uncovers the barriers of the underlying Frame DataBase (FDB) model [12]. The FDB model permits the use of composite data types, but not in any (arbitrary) depth, as the Nested Relational model [3, 11] permits. This was a key decision, since the FDB model should be accompanied with a data manipulation language, CUDL [5, 7], able to directly manipulation of the provided composite data structures.

5.1 The Modified Conceptual Model

In the following conceptual (ER) diagram (Figure 2), we introduce two attributes that characterize the participation of a physician (doctor) to the weak entity "Incident Operation". These are:

part_begin	representing the date and time that the participation of the doctor to the incident operation started,
part_end	representing the date and time that the participation of the doctor to the incident operation finished.

These attributes define the duration of participation of a doctor in some specific operation that is accomplished during some incident. In other words, the user requirements demand not only to document the participation of doctors in incident operations, but also to document when the doctor get involved and when he/she back off the operation.

5.2 Handling attributes relating a weak with a strong entity type

The following rule handles the translation of a relation between a weak and a strong (not identifying) entity that has attributes:

Rule 6. Binary relationships with cardinality ratio M:N, relating a weak entity type with some other (strong) entity type, having relationship attributes (assume k attributes), can be transformed to an extra subfield of the composite tag implementing the weak entity. This extra subfield will not have repetitions and should have unique values in the range of the whole set of frames of the strong identifying entity. Moreover another composite tag with k+1 subfields and permitting repetitions should be introduced in the related (strong) entity type. The relationship (k) attributes, together with the unique value of the extra subfield of the composite tag that implements the weak entity will combined to form the (k+1) subfield values of some repetition of the new composite tag of the related (strong) entity. This rule applies for the relationship between the weak entity type "Incident Operations" and the entity type Doctor.

5.3 The modified Incident and Doctor entity types

In contrast to Rule 4, Rule 6 requires the creation of identifiers for each instance of the weak entity of the relation and the storage of these identifiers into the frames (instances) of the strong (not identifying) related entity. This could be an alternative solution in case of a relation (between a weak entity type and a strong – not identifying – entity type) without attributes. This is the only solution when the relation (between the weak entity type and the strong – not identifying – entity type) has attributes. The following are the CAL structures of the strong – identifying – entity type "Incident" that hosts the weak entity type "Incident_operations" and the strong – not identifying – entity type "Doctor". The modifications (in relation to the previous versions of the same entity types) are emphasized.

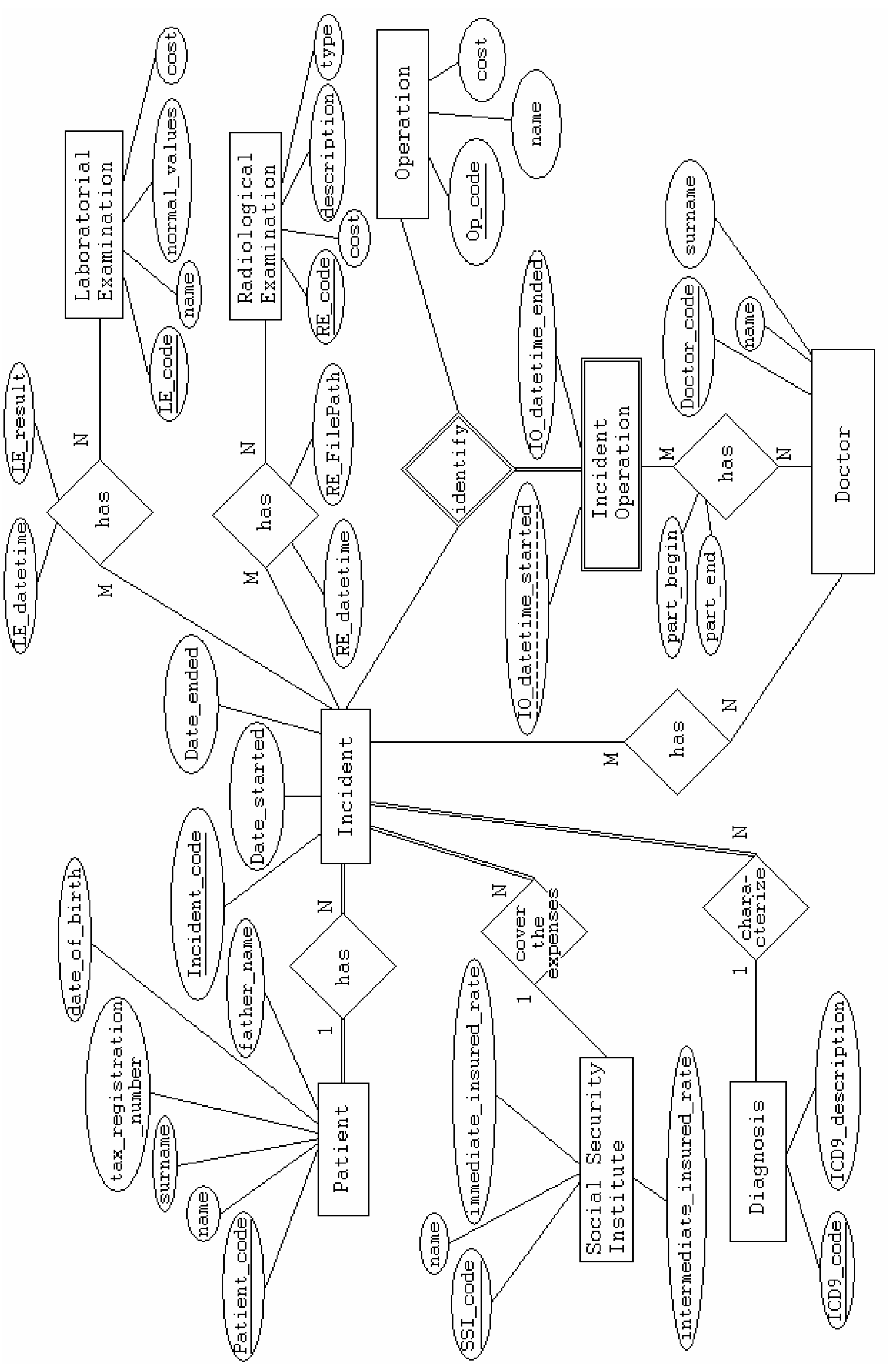

Figure 2: The modified conceptual model.

<u>Incident</u>
Incident_code, Date_started, Date_ended, Patient_code, SSI_code,
{Lab_examinations (LE_code, LE_datetime, LE_result)},
{Rad_examinations (RE_code, RE_datetime, RE_FilePath)},
{Incident_operations (Op_code, IO_datetime_started, IO_datetime_ended, **IO_ID**)},
{Incident_doctors}, ICD9_code.

<u>Doctor</u>
Doctor_code, name, surname,
{Operations_participated (IO_ID, part_begin, part_end)}.

Figure 3 portraits a CAL frame object (instance) for the entity "Incident".

Incident_code	S001
Date_started	13/5/2007
Date_ended	20/5/2007
Patient_code	A001
SSI_code	T001
Incident_doctors	I001 I002 I079
ICD9_code	574

Incident_ operations

Op_ code	IO_ datetime_started	IO_ Datetime_ended	IO_ID
E002	14/5/2007 13:35	14/5/2007 15:05	317
E015	16/5/2007 12:00	16/5/2007 13:00	318

Lab_ Examinations

LE_ code	LE_datetime	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

Rad_ Examinations

RE_code	RE_datetime	RE_FilePath
U/S Kidney	16/5/2007 12:00	\\FS1\RIS\ Uaz34.tif

Figure 3: An Incident frame object.

Figure 4 portraits some CAL frame objects (instances) for the entity type "Doctor":

5.4 Enforcement of data validation

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.

In CUDL the declaration of relationships between data structures is maintained in an FDB table (table of the FDB universal logical database schema) named Authority_Links [6]. The structure of this table is the following:

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

For example, the M:N relationship between the entity type "Incident" and the entity type "Doctor" is declared in the following way:

('Incident', 'Incident_doctors', null, 'Doctor', 'Doctor_code', null, 1)

Doctor_code	I001
name	Manos
surname	Grigoropoulos

Operations_participated

IO_ID	part_begin	part_end
317	14/5/2007 13:35	14/5/2007 15:05

(a)

Doctor_code	I005
name	Petet
surname	Antoniou

Operations_participated

IO_ID	part_begin	part_end
317	14/5/2007 13:35	14/5/2007 14:25

(b)

Doctor_code	I012
name	…
surname	…

Operations_participated

IO_ID	part_begin	part_end
318	16/5/2007 12:00	16/5/2007 13:00

(c)

Doctor_code	I032
name	…
surname	…

Operations_participated

IO_ID	part_begin	part_end
318	16/5/2007 12:00	16/5/2007 13:00

(d)

Doctor_code	I065
name	…
surname	…

Operations_participated

IO_ID	part_begin	part_end
317	14/5/2007 13:35	14/5/2007 15:05

(e)

Doctor_code	I100
name	…
surname	…

Operations_participated

IO_ID	part_begin	part_end
317	14/5/2007 14:25	14/5/2007 15:05
318	16/5/2007 12:00	16/5/2007 13:00

(f)

Figure 4: Some Doctor frame objects.

The last attribute in the previous row of the Authority_links table is the integer value of one (1). It defines a simple M:N relationship where the data implementing the relationship are kept in the M side. The whole range of values of the relationship_type attribute is provided in table 1.

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$_{\downarrow}$
12	Simple M:N, data implementing the relationship are kept in subfield in the N side	M:N/R$_{\downarrow}$
14	S(M):N, data implementing the relationship are kept in subfield in the N side	S(M):N/R$_{\downarrow}$
15	N:S(M), data implementing the relationship are kept in subfield in the N side	N:S(M)/L$_{\downarrow}$
20	Authority control over Tag	A/T
21	Authority control over Subfield	A/S

Table 1: The range of values for the attribute relationship_type**.**

According to table 1 and based on the early solution where there are no attributes for the relationship between "Incident operations" (the operations that a patient undertook during the period of an incident) and "Doctors", the relationship is declared as:

('Incident', 'Incident_operstions', 'IO_doctors', 'Doctor', 'Doctor_code', null, 3)

In case that the relationship between "Incident operations" and "Doctors" has attributes and according to the modified CAL entity types, the relationship is declared as:

('Incident', 'Incident_operstions', 'IO_ID', 'Doctor', 'Operations_participated', 'IO_ID', 14)

6. Conclusions

So far, the design of an application having a relational data repository mainly required the decomposition of the real world structures into very simple attributes, the composition of a logical schema with naive relational structures and the formation of query and manipulation (SQL) statements based on the logical schema. There is a need for a database query and manipulation language, like CUDL, able to handle directly composite entities of more abstract data models that offer composite / complex data types expressing real world entities (without transforming them to a relational logical schema). The types used in CUDL allow a database designer / developer to express the structures of his application with types that are very close to the ER entity types, while ER entity types can be directly transformed to CAL entity types. Thus, the designers and developers can be concentrated with the business logic of their applications, instead of wasting time for the expression of statements that manipulate naive database structures.

The set of rules for transforming (the relationship types of) an ER diagram to CAL, so that the CUDL can be utilized to manipulate the resulting high level entities, that was presented in [8], was extended in this paper with an extra rule about the translation of a relation between a weak and a strong (not identifying) entity that also has attributes. This rule applies in a specific subcase of relation between entity types that uncovers the barriers of the underlying Frame DataBase (FDB) model [12] (the FDB model permits the use of composite data types, but not in any depth).

Possible future research directions include

- completing the analysis and design process that consists in the quadruple “ER, CAL, FDB, physical level” by providing a physical model and a transformation of the FDB to this model,
- implementing a database machine storing data based on this physical model and being able to process CUDL queries,
- designing algorithms for optimized processing of CUDL queries,
- evaluating this machine in terms of performance and suitability for development of complex applications.

References

[1] Cai. J., Johnson, S., Hripcsak, G.: Generic Data Modeling for Home Telemonitoring of Chronically Ill Patients. In: American Medical Informatics Association - Annual Symposium 2000 (AMIA 2000), pp. 116--120. Los Angeles, CA (2000)

[2] Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, 3rd Edition. Addison Wesley Publishing Company, Reading, Mass. (2000)

[3] Fischer P.C., Van Gucht D.: Determining when a Structure is a Nested Relation. In: 11th Int. Conf. on Very Large DataBases (VLDB 1985), pp. 171--180. Morgan Kaufmann, Stockholm, Sweden (1985)

[4] Johnson, S.B.: Generic data modeling for clinical repositories. Journal of American Medical Informatics Association 3 (5), 328--339 (1996)

[5] Karanikolas, N.N., Nitsiou, M., Yannakoudakis, E.J., Skourlas, C.: CUDL language semantics, liven up the FDB data model. In: 11th East-European Conf. on Advances in Databases and Information Systems (ADBIS 2007), local proceedings, pp. 1--16. Technical Univ. of Varna, Varna, Bulgaria (2007)

[6] Karanikolas, N.N., Nitsiou, M., Yannakoudakis, E.J.: CUDL Language Semantics: Authority Links. In: 12th East-European Conf. on Advances in Databases and Information Systems (ADBIS 2008), pp. 123--139. Tampere Univ. of Technology, Pori, Finland (2008)

[7] Karanikolas, N. N., Nitsiou, M., Yannakoudakis, E. J., Skourlas, C: CUDL Language Semantics: Updating Data. Journal of Systems and Software 82 (6), 947-962 (2009) doi:10.1016/j.jss.2008.12.031

[8] Karanikolas N. and Vassilakopoulos M., Conceptual Universal Database Language: Moving Up the Database Design levels, *Proc. of ADBIS'09,* LNCS 5739, Riga (2009), *330-346.*

[9] Nadkarni P.M.: Clinical Patient Record Systems Architecture: An Overview. Journal of Postgraduate Medicine 46 (3), 199--204 (2000)

[10] Nadkarni P.: An introduction to entity-attribute-value design for generic clinical study data management systems. Presentation in: National GCRC Meeting. Baltimore, MD (2002)

[11] Schek H.J., Pistor P.: Data Structures for an Integrated Data Base Management and Information Retrieval System. In: 8th Int. Conf. on Very Large DataBases (VLDB 1982), pp. 197--207. Morgan Kaufmann, Mexico City, Mexico (1982)

[12] Yannakoudakis E.J., Tsionos C.X., Kapetis C.A.: A new framework for dynamically evolving database environments. Journal of Documentation 55 (2), 144--158 (1999)

[13] Yannakoudakis E.J., Diamantis I.K.: Further improvements of the Framework for Dynamic Evolving of Database environments. In: 5th Hellenic – European Conf. on Computer Mathematics and its Applications (HERCMA 2001). Athens, Greece (2001)

[14] Yannakoudakis E.J., Nitsiou M., Skourlas, C., Karanikolas, N.N.: Tarski algebraic operations on the frame database model (FDB). In: 11th Panhellenic Conf. in Informatics (PCI 2007), pp. 207--216. New Technologies Publications, Patras, Greece (2007).