MatBase – a Tool for Transparent Programming while Modelling Data at Conceptual Levels

Christian Mancas

Department of Mathematics and Computer Science, Ovidius University, Constanta, Romania

Abstract

MatBase is a prototype intelligent data and knowledge base management system based on the Relational, Entity-Relationship, and (Elementary) Mathematical Data Models, having two current versions (MS SQL Server and C#, MS Access and VBA). Users may work with it only at one or any combination of these conceptual levels, without any programming knowledge (be it SQL, C#, VBA, etc.), to create, populate, update, and delete databases and corresponding management software applications. The paper introduces the MatBase architecture and the principles used to transparently program while modelling data at these three conceptual levels with this tool. A real-life example illustrates them.

Keywords

Conceptual Data Modelling, Automatic Code Generation, Relational Constraints, Non-relational Constraints, DBMS Engine Architectures, The (Elementary) Mathematic Data Model, MatBase

1. Introduction

MatBase [8, 9, 10, 11, 12, 13, 14] is a prototype intelligent Knowledge and Database Management System (KDBMS) built on top of an existing relational DBMS (RDBMS) and based on both the Relational Data Model (RDM) [1, 4, 8], the Entity-Relationship one (E-RDM) [3, 8, 19], and the (Elementary) Mathematical one ((E)MDM) [8, 9, 10, 11, 12], which also embeds Datalog$\neg$ [1, 9]: its users may define, update, and delete database (db) schemas in any of these three formalisms, which *MatBase* is automatically translating into the other two ones. Moreover, *MatBase* also automatically builds db software applications for managing their data instances plausibility not only for the six RDM constraint types provided by any RDBMS, but also for the 61 ones of the (E)MDM, as well as for querying the managed db instances, mathematically too. Why was it architectured, designed, and developed like that? The answer is two-folded.

First of all, because dbs should be architectured and designed not at the RDM level, which is almost purely syntactic, but at higher level ones and, not only in our opinion, the best first one in such a data modeling hierarchy is the E-RDM – probably the only data model that can be also understood by our customers. Then, although there are algorithms to translate E-R diagrams (E-RDs) to relational db schemes, and even directly to some RDBMS versions ones [8], as E-RDM is still too poor as provided constraint types, it is highly advisable to first refine E-RDs, by adding all existing constraints in the modeled sub-universe, for which we are advocating the use of the (E)MDM.
Secondly, in general, humanity was and will always be designing and using more and more ad-

vanced tools for coping with complexity, in order to hide as many tedious aspects of life as possible, and to focus on problem understanding and solving at the highest possible conceptual level in its corresponding eras.

For example, even the biggest fans of the Linux operating system are turning, even if slowly, to graphical user interfaces (GUIs, e.g. [17]). For example, even the biggest fans of SQL programming are first using RDBMS GUIs when specifying and/or modifying db schemes and/or queries to be run against them and only then, when needed, they manually improve the automatically generated SQL scripts. For example, space crafts, airplanes, vehicles, robots, buildings, human tissues, organs, bones, etc., as well as so many other things are architectured and designed using AutoCAD [2] and alike software tools (and some of them even directly manufactured with 3D printers [16]).

Current versions of *MatBase* are implemented in MS Access (for students and small dbs) and in MS C#.Net and SQL Server (for professionals and/or large dbs). This paper focuses on how *MatBase* automatically turns E-RDM, (E)MDM, and RDM data models into programs, thus making possible for non-programmers to design, query, and obtain db management software applications only working at conceptual data modeling levels.

1.1. Related Work

Probably the erwin Data Modeler [5] is the most used and well-known RDBMS based on the E-RDM, among dozens of others. Advanced RDBMSes like MS SQL Server, Oracle, and IBM DB2 also provide both data modelers (MS Entity Data Model Designer (based on the MS Entity Framework) [18], Oracle SQL Developer Data Modeler [6], IBM InfoSphere Data Architect [7]) and GUI-type querying interfaces (MS Design View, Oracle Query Builder, IBM QBE). *MatBase* adds to their corresponding facilities mainly its (E)MDM and Datalog¬ interfaces, which are the most powerful of all of them.

1.2. Paper Outline

The following two sections introduce the *MatBase* architecture and the principles used to transparently program while modeling data at conceptual levels. Section four illustrates them with a real-life example. The paper ends with conclusion and references.

2. *MATBASE* ARCHITECTURE

Figure 1 shows the overall *MatBase* simplified architecture, which is a standard 4-tier one, whose tiers are (in the top-down order) the following: GUI, business logic (BL), ActiveX Data Objects (ADO), and db (DB).

The MS *Access* version DB is composed of two shared dbs, namely MatBaseDB.accdb and GeographyDB.accdb (stored on a file server folder, identified through network mapping as virtual logic drive V:), and each workstation storing eight other dbs, namely MatBaseTmp.accdb, GeographyTmp.accdb, StocksDB.accdb, StocksTmp.accdb, BookstoreDB.accdb, BookstreTmp.accdb, UserDB.accdb, and UserTmp.accdb (stored in a folder declared, through *subst*, as being virtual logic drive U:).

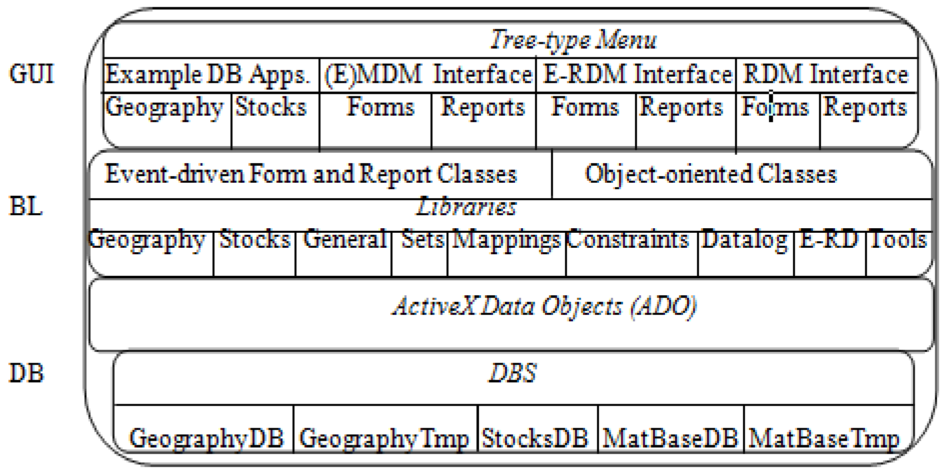

Figure 1. *MatBase* overall architecture

MatBaseDB contains the *Matbase*'s metacatalog (storing metadata on the managed dbs, tables, constraints, etc.) and knowledge base (storing data on coherence and minimality of constraint sets, Datalog¬ inference rules, object constraint types ones, etc.). As *MatBase* also provides four db application examples (*Geography*, shareable, *Stocks*, *Bookstore*, and *User*) there are also db files storing their data. All needed temporary tables are stored in corresponding Tmp dbs.
In its MS SQL Server versions, all these 5 fundamental dbs are stored on a server instance, whereas all temporary ones are stored in the corresponding system tempDB.

ADO is the (de facto industry standard) middleware between BL and DB, completely transparent to programmers in the .net and SQL Server *MatBase* version, carrying SQL statements as strings from BL to DB and returning error codes and selected data. In the Access version, ADO is also explicitly used, as, for example, there is no CHECK constraint in its SQL, and no triggers or stored procedures either.

For its both versions, BL is made of object-oriented classes (most of them also event-driven, associated to forms and reports) and libraries grouping methods commonly used by at least two classes. Every db application has its own library (e.g. *Geography*, *Stocks*, *Bookstore*, etc.). *MatBase*'s core has several specialized ones: *Constraints*, *Datalog*, *ERD*, *General*, *Mappings*, *Sets*, and *Tools*, with *General* including methods, variables, and constants commonly used by at least two other libraries.

For example, the *Constraints* one includes parameterized Boolean functions for enforcing object constraints (*enforceObjConstraint*), dyadic relation reflexivity (*enforceDRReflex*), irreflexivity (*enforceDRIrreflex*), asymmetry (*enforceDRAsymm*) etc. Whenever the current data passed as parameters satisfies the corresponding constraint, these functions return *False*, otherwise they return *True* (following the MS event-driven methods' *Cancel* parameter conventions).

GUI includes a tree-like menu, forms, and reports, all of them carefully designed, as user-friendly as possible. Each fundamental db table has at least one standard associated form, automatically generated, which manages corresponding data management (i.e. inserts / updates / deletes).

3. Transparent Programming while Modelling Data at Conceptual Levels

Object, system, and dynamically enumerated value sets are implemented in *MatBase*, just like in any RDBMS, as tables, the rest of the value ones as data types, whereas computed ones as views. Mappings are implemented as table/view columns. The six relational ones (not null, default value, domain/range, unique and foreign keys, as well as tuple/check) are automatically enforced in rdbs whenever they are asserted by users; *MatBase* implements them using its underlying RDBMS corresponding mechanisms [13]. Any other (non-relational) constraint type needs programming, with either extended SQL triggers or high-level programming languages embedding SQL event-driven methods, which are automatically generated by *MatBase*. Datalog¬ programs are stored by *MatBase* into its meta-catalogue together with their corresponding relational algebra (RA) equation systems (obtained through syntax-directed translation) and their instances are computed by using the least fixpoint computational semantics [1, 9].

Note that advanced commercial RDBMS (e.g. MS SQL Server, Oracle, IBM DB2), according to the ANSI 1999 object-oriented SQL standard [15] and all later ones, are only providing a subset of Datalog, namely recursive SELECT queries.

Whenever enforcing of a non-relational constraint is possible (i.e. it would not violate coherence or minimality of the corresponding constraint set and the current db instance satisfies it), *MatBase* injects in the corresponding *BeforeUpdate* / *Validating* event-driven method associated to the corresponding set or mapping (in the object-oriented class(es) of the form(s) associated with the table corresponding to the domain set) an assignment of type *Cancel* = call to the corresponding Boolean function from *Constraints*; therefore, if the returned value is *False*, then the corresponding data update is accepted and saved in the db; otherwise, it is rejected with a corresponding context-dependent error message and users are invited to modify it or cancel the current request.

3.1. E-RD Modelling

E-RDs can be drawn, saved, modified, and deleted, just like in the erwin Data Modeler. The only main difference is that *MatBase* is also creating, updating, and deleting not only corresponding relational dbs, but also corresponding (E)MDM schemas of them, as well as forms for managing data built on every fundamental db table. Minor, but important differences are consequences of its variant of E-RDs [8], which represents functional relationships as arrows instead of diamonds, allows for *n*-ary relationships, $n > 1$, as well as for relationship hierarchies.

3.2. (E)MDM Modelling

Both object, value, and computed sets, mappings defined on and taking values from them, constraints, and Datalog¬ programs may be added, modified, and dropped. Corresponding E-RDs may be generated and saved. For complex db schemes, users may choose a central object set and specify a natural radius to generate only the corresponding sub-diagram. For each fundamental table, a standard form (that can be then modified) for managing corresponding instances is automatically generated (other forms may be also added any time after). Set elements (including mapping images' ones) may be inserted, updated, queried, and deleted through these forms. Code is automatically injected in these forms' associated classes for enforcing non-relational constraints.

Moreover, db instances can be mathematically queried too (using the semi-naïve algebra of sets, relations, and functions operators, as well as the first order predicate calculus ones), as *MatBase* translates them into corresponding SQL queries (e.g. replacing function compositions with joins, function Cartesian products with column concatenation, etc.) and then passes them for execution to the corresponding host RDBMS.

3.3. RDM Modelling

Dbs, tables, relational constraints, views, and indexes, as well as triggers, procedures, functions, sequences, etc. can be added, populated, queried, updated, and dropped. This can be done either from scratch, or upon dbs created through the other two interfaces. Corresponding (E)MDM schemes are automatically created, updated, and dropped. For each fundamental table, a standard form (that can be then modified) for managing corresponding instances is generated (other forms may be also added then by users).

Moreover, queries can be graphically designed, updated, and saved: *MatBase* is automatically generating the corresponding SQL code for its host RDBMS.

4. A REAL-LIFE EXAMPLE

Let us consider the sub-universe of countries and continents, for which, for simplicity, only names and neighbouring relations are of interest. The corresponding business rules are the following:

- Countries have unique names that are compulsory.
- Continents have unique names that are compulsory.
- Countries belong to the continent where their capital is located.
- Some countries may span over several continents (e.g. Russia and Turkey).
- No country or continent may be neighbor to itself.
- Whenever countries or continents *x* and *y* are neighbor to each other, then *y* and *x* are neighbors too.
- Any neighbor countries may belong to either same continent or neighbor continents.

Figure 2 shows the corresponding E-RD and Figure 3 its associated restriction set [8]. Note that both *NEIGHBOR_CONTINENTS* and *NEIGHBOR_COUNTRIES* are, both mathematically and in reality, symmetric, but, from the conceptual data modelling point of view are (apparently paradoxically) asymmetric: we only have three choices (not enforcing these constraints, enforcing symmetry, and enforcing asymmetry); not enforcing them is the worst, as implausible instances are possible (e.g. storing both <U.S.A., Canada> and <Canada, U.S.A.>, but only <U.S.A., Mexico>, without its dual <Mexico, U.S.A.> as well); as no user would be happy to be forced to always insert symmetrical pairs, enforcing symmetry would require writing code to immediately insert, update, and delete corresponding symmetric pairs; enforcing asymmetry is the best solution from many points of view: code needs to be written only for inserts and updates (to prevent storing symmetric pairs); no information is lost, as symmetric pairs can be simply computed whenever needed with a very simple SELECT SQL query; corresponding table instances are half the size as compared to enforcing symmetry.

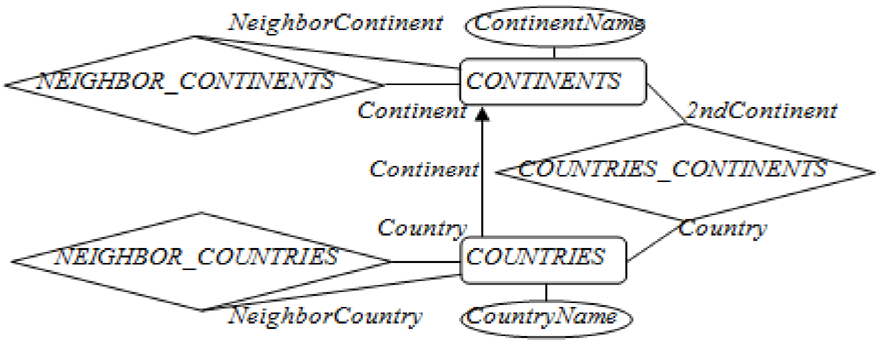

Figure 2. E-RD for the simplified countries and continents sub-universe

Also note that, although not explicitly stated as a business rule, the (function diagram anti-commutativity type) constraint “No country may belong twice to a same continent.” (stemming from the first db axiom “In any db, any data should be stored only once.”) is needed as well: not enforcing it would allow implausible instances like storing Europe for Russia in *COUNTRIES* as its continent and also storing Europe for Russia in *COUNTRIES_CONTINENTS*, alongside with Asia.

Applying the Algorithms A1 and A2 from [8], this E-R data model can be translated into the (E)MDM schema shown in Figure 4.

Applying then the Algorithm A7 from [8], this (E)MDM schema can be translated into the rdb schema shown in Figure 5 (together with a plausible valid instance of it) and the associated non-relational constraint set shown in Figure 6.

CONTINENTS
1.*Max. cardinality*: 8
2.*Ranges ContinentName*: ASCII(16)
3.*Mandatory*: *ContinentName*
4.*Uniquenesses*: *ContinentName*

NEIGHBOR_CONTINENTS
1.*Max. cardinality*: 8
2.*Ranges*:
3.*Mandatory*: *Continent, NeighborContinent*
4.*Uniquenesses*:
Continent • NeighborContinent
5.*Other restrictions*: *NEIGHBOR_ CONTINENTS* irreflexive, asymmetric

COUNTRIES
1.*Max. cardinality*: 256
2.*Ranges CountryName*: ASCII(128)
3.*Mandatory*: *CountryName, Continent*
4.*Uniquenesses*: *CountryName*

NEIGHBOR_COUNTRIES
1.*Max. cardinality*: 2048
2.*Ranges*:
3.*Mandatory*: *Country, NeighborCountry*
4.*Uniquenesses*:
Country • NeighborCountry
5.*Other restrictions*: *NEIGHBOR_ COUNTRIES* irreflexive, asymmetric

COUNTRIES_CONTINENTS
1.*Max. cardinality*: 8
2.*Ranges*:
3.*Mandatory*: *Country, 2ndContinent*
4.*Uniquenesses*: *Country • 2ndContinent*
5.*Other constraints*: No country may belong twice to a same continent.
Any neighbor countries may belong to either same continent or neighbor continents.

Figure 3. Restriction set associated to the E-RD in Figure 2

4.1. Using the E-RDM *MatBase* Interface to Implement the E-RD from Figure 2

First, users should open a blank E-RD, then draw in it the E-RD from Figure 2, and finally save it in a new db (say CountriesDB). *MatBase* is then immediately creating the corresponding db with a relational scheme similar to the one in Figure 5, except that both *ContinentName* and *CountryName* will be declared as VARCHAR(255) (i.e. maximum short text), without not nulls and without unique keys, except for the surrogate primary ones *x* (all declared as long integers), in *CONTINENTS* and *COUNTRIES*. This is done by generating and then executing corresponding SQL scripts against its host RDBMS. Moreover, it will store in its meta-catalogue not only the new db and its E-RD data, but also the corresponding (E)MDM one, and will create standard management forms for all five tables.

x : *CONTINENTS* ↔ autonumbering(1)
ContinentName : *CONTINENTS* ↔ ASCII(16), total
x : *COUNTRIES* ↔ autonumbering(3)
CountryName : *COUNTRIES* ↔ ASCII(128), total
Continent : *COUNTRIES* → *CONTINENTS*, total
NEIGHBOR_CONTINENTS = (*Continent* → *CONTINENTS*,
NeighborContinent → *CONTINENTS*) irreflexive, asymmetric
x : *NEIGHBOR_CONTINENTS* ↔ autonumbering(1)
NEIGHBOR_COUNTRIES = (*Country* → *COUNTRIES*,
NeighborCountry → *COUNTRIES*) irreflexive, asymmetric
x : *NEIGHBOR_COUNTRIES* ↔ autonumbering(4)
COUNTRIES_CONTINENTS = (*Country* → *COUNTRIES*,
2ndContinent → *CONTINENTS*)
x : *COUNTRIES_CONTINENTS* ↔ autonumbering(1)
C5 (No country may belong twice to a same continent.): (∀*x*∈*COUNTRIES*)
(∀*y*∈*COUNTRIES_CONTINENTS*)(x = *Country*(*y*) ⇒ *Continent*(*x*) ≠ *2ndContinent*(*y*))
C6 (Any neighbour countries may belong to either same continent or neighbour continents.):
(∀*x*∈*NEIGHBOR_COUNTRIES*) (*Continent*(*Country*(*x*)) = *Continent* (*NeighborCountry*(*x*)) ∨
(∃z∈*COUNTRIES_CONTINENTS*) (*Continent*(*Country*(*x*)) = *2ndContinent*(z) ∧ *NeighborCountry*(*x*) =
Country(z) ∨ *Continent*(*NeighborCountry*(*x*)) = *2ndContinent*(z) ∧ *Country*(*x*) = *Country*(z) ∨
(∃*y*∈*NEIGHBOR_CONTINENTS*) (*Continent*(*Country*(*x*)) = *Continent*(*y*) ∧ *Conti-*
nent(*NeighborCountry*(*x*)) = *NeighborContinent*(*y*) ∨ *Continent*(*Country*(*x*)) = *NeighborContinent*(*y*) ∧
Continent(*NeighborCountry*(*x*)) = *Continent*(*y*) ∨ *Continent*(*Country*(*x*)) = *Continent*(*y*) ∧ *2ndConti-*
nent(z) = *NeighborContinent*(*y*) ∧ *Country*(z) = *NeighborCountry*(*x*) ∨ *Continent*(*NeighborCountry*(*x*)) =
Continent(*y*) ∧ *2ndContinent*(z) = *NeighborContinent*(*y*) ∧ *Country*(z) = *Country*(*x*))))

Figure 4. The (E)MDM scheme corresponding to the E-R data model from Figures 2 and 3

***CONTINENTS*(x, *ContinentName*)**

x	***ContinentName***
auton(1)	*ASCII(16)*
NOT NULL	*NOT NULL*
1	Europe
2	Asia
3	North America
4	South America
5	Africa
6	Australia
7	Oceania
8	Antarctica

***COUNTRIES*(x, *CountryName*)**

x	***Country-Name***	***Continent***
au-ton(3)	*ASCII(128)*	*CONTI-NENTS.x*
NOT NULL	*NOT NULL*	*NOT NULL*
1	Romania	1
2	Slovenia	1
3	Ukraine	1
4	Hungary	1
5	Russia	1

***NEIGHBOR_CONTINENTS*(x, *Continent* • *NeighborContinent*)**

x	***Continent***	***NeighborContinent***
auton(1)	*CONTINENTS.x*	*CONTINENTS.x*
NOT NULL	*NOT NULL*	*NOT NULL*
1	1	2
2	2	5
3	3	4

***NEIGHBOR_COUNTRIES*(x, *Country* • *NeighborCountry*)**

x	***Country***	***NeighborCountry***
auton(4)	*COUNTRIES.x*	*COUNTRIES.x*
NOT NULL	*NOT NULL*	*NOT NULL*
1	1	3
2	1	4
3	2	4
4	3	4
5	3	5

***COUNTRIES_CONTINENTS*(x, *Country* • *2ndContinent*)**

x	***Country***	***2ndContinent***
auton(1)	*COUNTRIES.x*	*CONTINENTS.x*
NOT NULL	*NOT NULL*	*NOT NULL*
1	5	2

Figure 5**.** Relational db schema corresponding to the (E)MDM scheme in Figure 4 and a plausible valid instance of it

C1: *NEIGHBOR_CONTINENTS* irreflexive
C2: *NEIGHBOR_CONTINENTS* asymmetric
C3: *NEIGHBOR_COUNTRIES* irreflexive
C4: *NEIGHBOR_COUNTRIES* asymmetric
C5 (No country may belong twice to a same continent.): (∀*x*∈*COUNTRIES*)
(∀*y*∈*COUNTRIES_CONTINENTS*)(*x* = *Country*(*y*) ⇒ *Continent*(*x*) ≠ *2ndContinent*(*y*))
C6 (Any neighbour countries may belong to either same continent or neighbour continents.):
(∀*x*∈*NEIGHBOR_COUNTRIES*) (*Continent*(*Country*(*x*)) = *Continent* (*NeighborCountry*(*x*)) ∨
(∃*z*∈*COUNTRIES_CONTINENTS*) (*Continent*(*Country*(*x*)) = *2ndContinent*(*z*) ∧ *NeighborCountry*(*x*) =
ountry(*z*) ∨ *Continent*(*NeighborCountry*(*x*)) = *2ndContinent*(*z*) ∧ *Country*(*x*) = *Country*(*z*) ∨
(∃*y*∈*NEIGHBOR_CONTINENTS*) (*Continent*(*Country*(*x*)) = *Continent*(*y*) ∧ *Conti-*
nent(*NeighborCountry*(*x*)) = *NeighborContinent*(*y*) ∨ *Continent*(*Country*(*x*)) = *NeighborContinent*(*y*) ∧
ontinent(*NeighborCountry*(*x*)) = *Continent*(*y*) ∨ *Continent*(*Country*(*x*)) = *Continent*(*y*) ∧ *2ndConti-*
nent(*z*) = *NeighborContinent*(*y*) ∧ *Country*(*z*) = *NeighborCountry*(*x*) ∨ *Continent*(*NeighborCountry*(*x*)) =
ontinent(*y*) ∧ *2ndContinent*(*z*) = *NeighborContinent*(*y*) ∧ *Country*(*z*) = *Country*(*x*))))

Figure 6. Non-relational constraint set associated to the rdb scheme in Figure 5

4.2. Using the E-RDM *MatBase* Interface to Implement the E-RD from Figure 2

First, users should open a blank E-RD, then draw in it the E-RD from Figure 2, and finally save it in a new db (say CountriesDB). *MatBase* is then immediately creating the corresponding db with a relational scheme similar to the one in Figure 5, except that both *ContinentName* and *CountryName* will be declared as VARCHAR(255) (i.e. maximum short text), without not nulls and without unique keys, except for the surrogate primary ones *x* (all declared as long integers), in *CONTINENTS* and *COUNTRIES*. This is done by generating and then executing corresponding SQL scripts against its host RDBMS. Moreover, it will store in its meta-catalogue not only the new db and its E-RD data, but also the corresponding (E)MDM one, and will create standard management forms for all five tables.

4.3. Using the (E)MDM *MatBase* Interface to Refine the DB as per Figure 4

To refine the newly created db according to Figure 4, users should then open it in the (E)MDM interface and do the following:

6. Modify 256 from the codomains of *ContinentName* to 16 and of *CountryName* to 128 in the form *FUNCTIONS*.
7. Click on *Total?* for *ContinentName*, *CountryName*, and *Continent* from *COUNTRIES* in the form *FUNCTIONS*.
8. Click on *Key?* for *ContinentName* and *CountryName* in the form *FUNCTIONS*.
9. Click on *Irreflexive?* and *Asymmetric?* for *NEIGHBOR_COUNTRIES* and *NEIGHBOR_CONTINENTS* in the form *RELATIONS*.

Add the following two lines in the form *OBJ_CONSTRAINTS*, by selecting for both of them the db *CountriesDB* from the combo-box *DB*, typing C5 and C6, respectively, in the text box *ConstraintName*, and, optionally, copying from this paper and pasting in the *Semantics* text box the corresponding descriptions, and typing in the *Constraint* text box (or, much simpler, copying them from this paper, pasting them in a temporary .doc file, replacing everywhere ∀ with “for any”, ∃ with “there is”, ∈ with “ in “, ⇒ with “=>”, ≠ with “!=”, ∨ with “OR”, and ∧ with

"AND", and then copying the resulted temporary texts and pasting them into the corresponding text box *Constraint*), respectively:

- (for any x in COUNTRIES) (for any y in COUNTRIES_CONTINENTS)(x = Country(y) => Continent(x) != 2ndContinent(y))
- (for any x in NEIGHBOR_COUNTRIES) (Continent(Country(x)) = Continent (NeighborCountry(x)) OR (there is z in COUNTRIES_CONTINENTS) (Continent(Country(x)) = 2ndContinent(z) AND NeighborCountry(x) = Country(z) OR Continent(NeighborCountry(x)) = 2ndContinent(z) AND Country(x) = Country(z) OR (there is y in NEIGHBOR_CONTINENTS) (Continent(Country(x)) = Continent (*y*) AND Continent(NeighborCountry(x)) = NeighborContinent(y) OR Continent (Country(x)) = NeighborContinent(y) AND Continent(NeighborCountry(x)) = Continent(y)) OR Continent(Country(x)) = Continent(y) AND 2ndContinent(z) = NeighborContinent(y) AND Country(z) = NeighborCountry(x) OR Continent (NeighborCountry(x)) = Continent(y) AND 2ndContinent(z) = NeighborContinent(y) AND Country(z) = Country(x))))

For 1 to 3 above, *MatBase* will automatically generate and execute corresponding SQL scripts to alter the db scheme accordingly. For 4, it will inject needed VBA / C# code in the event-driven methods attached to the events *Form_BeforeUpdate* / *Validating* in the classes attached to the forms built upon NEIGHBOR_COUNTRIES and NEIGHBOR_CONTINENTS for calling the methods *enforceDRIrreflex* and *enforceDRAsymm* of the *Constraints* library with the corresponding parameters. Similarly, for 5 it will inject code into the methods attached to the events *Form_BeforeUpdate* / *Validating* and *Form_Delete* in the classes attached to the forms built upon COUNTRIES_CONTINENTS and NEIGHBOR_CONTINENTS, as well as into the ones attached to the events *Form_BeforeUpdate* / *Validating* in the classes attached to the forms built upon NEIGHBOR_COUNTRIES for calling the method *enforceObjConstraint* of the *Constraints* library with the corresponding parameters.

The whole CountriesDB may be defined from the scratch in this interface (but this is more tedious) and then obtain the E-RD from Figure 2 by exporting it to the E-RDM.

Moreover, after populating it with data, users might use the *MatBase* (E)MDM interface for querying this db as well. For example, in order to get the "deciphered" instance of the NEIGHBOR_CONTINENTS, they might type in the *Query* text box of the *Queries* form "ContinentName(Continent), ContinentName(NeighborContinent)" (the *MatBase* syntax for the math compound functions Cartesian product ContinentName ° Continent • ContinentName ° NeighborContinent), which will be automatically translated by *MatBase* into the SQL statement:

```
SELECT CONTINENTS.ContinentName, CONTINENTS1.ContinentName
FROM (NEIGHBOR_CONTINENTS INNER JOIN CONTINENTS
ON NEIGHBOR_CONTINENTS.Continent = CONTINENTS.x)
INNER JOIN CONTINENTS AS CONTINENTS1
ON NEIGHBOR_CONTINENTS.NeighborContinent = CONTINENTS1.x
```

that will then be passed to the host RDBMS for execution.

4.4. Using the RDM *MatBase* Interface to Manage, Finetune, and Query the DB

Advanced users might finally use the RDM *MatBase* interface for finetuning and/or querying the db *CountriesDB* in SQL. For example, in order to speedup both enforcement of constraint *C6* and queries joining, filtering, and/or grouping on these columns, they may add indexes on *Country* and *NeighborCountry* from *NEIGHBOR_COUNTRIES*, as well as on *Continent* from *COUNTRIES*.

Moreover, except for the non-relational constraints, *CountriesDB*, like any other rdb, can be created, populated, modified, queried, and dropped from this interface as well. Any modification is automatically propagated by MatBase to the corresponding E-RD and (E)MDM schemes. Dropping a relational db scheme, if confirmed by users, is also dropping the E-RD and (E)MDM corresponding ones.

5. CONCLUSIONS AND FURTHER WORK

We have presented the simplified overall architecture and principles of design and implementation of the current two versions (for MS Access / VBA and SQL Server / C#) of *MatBase*, a prototype intelligent KDBMS providing data conceptual modeling in both E-RDM, (E)MDM, and RDM. We have illustrated with a real-life example how *MatBase* automatically generates code for creating, maintaining, querying, and dropping dbs, as well as software applications for their instances' management.

Compared to the other existing E-RDM tools, *MatBase*, first of all, using its knowledge base, is more useful to users as, for example, when implementing tables corresponding to relationships, is also automatically adding not null constraints for all of their roles (as they correspond to the math canonical Cartesian projections); moreover, for binary ones, it also adds the semantic (candidate) key made of the columns corresponding to the two roles (e.g. the three relationships from Section 4).

Secondly, which is much more important, *MatBase* also provides an (E)MDM interface that, besides offering Datalog¬'s full power, lets users model data at the math level as well, using the full power of the semi-naïve algebra of sets, relations, and functions and of the first order predicate calculus. Thirdly, *MatBase* also automatically generates software application forms for all db fundamental tables, in which it injects needed code for enforcing non-relational constraints as well.

Moreover, *MatBase* also provides wizards for assisting detection of all keys of any set [12], of all cycles in E-RDs [14], of all non-relational constraints associated to these cycles [11], as well as for guaranteeing constraint sets coherence [10].

Consequently, *MatBase* makes programming (be it in SQL or C# / VBA / ADO) completely transparent to its users, letting them create, populate, maintain, query, and drop dbs only at higher conceptual levels, while guaranteeing the plausibility of their instances. For them, conceptual data modeling is (transparent) db programming too.

MatBase is successfully used both in our lectures and labs on Databases (for undergraduate students), as well as for Advanced Databases (for M.Sc. postgraduate students) of the Mathematics and Computer Science Departments and by two Romanian IT companies developing db software applications for many U.S. and European customers in the Fortune 100 ones.

Using *MatBase* is especially beneficial to undergraduate students, as during their logic courses and labs they are generally not exposed to complex first order predicate formulas, at least like C6 from Section 3. Maximum benefits are, however, obtained by IT company users whose productivity is significantly boosted.

Further work is planned for designing and implementing a web *MatBase* version from the C# and SQL Server one, by using ASP.NET as well.

REFERENCES

[1] Abiteboul, S., Hull, R., Vianu, V. (1995) Foundations of Databases. Addison-Wesley, Reading, MA.

[2] Ascent Center for Technical Knowledge (2019) AutoCAD 2020 Fundamentals. Ascent, Charlottesville, VA.

[3] Chen, P. P. (1976) The entity-relationship model: Toward a unified view of data. ACM Transactions on Database Systems 1(1), 9–36.

[4] Codd, E. F. (1970) A relational model for large shared data banks. CACM 13(6), 377–387.

[5] DeAngelis, M. C. (2000) Data Modeling with ERwin. Sams Publishing, Indianapolis, IN.

[6] Helskyaho, H. (2015) Oracle SQL Developer Data Modeler for Database Design Mastery. Oracle Press – Mc Grow-Hill Education, New York, NY.

[7] IBM Corp. (2012) Smarter Modeling of IBM Master Data Management Solutions. IBM Redbooks, New York, NY.

[8] Mancas, C. (2015) Conceptual Data Modeling and Database Design: A Completely Algorithmic Approach. Volume I: The Shortest Advisable Path. Apple Academic Press / CRC Press (Taylor & Francis Group), Waretown, NJ.

[9] Mancas, C. (2019, in press) Conceptual Data Modeling and Database Design: A Completely Algorithmic Approach. Volume II: Refinements for an Expert Path. Apple Academic Press / CRC Press (Taylor & Francis Group), Waretown, NJ.

[10] Mancas, C. (2018) *MatBase* constraint sets coherence and minimality enforcement algorithms. In: Benczur, A., Thalheim, B., Horvat, T. (eds.) ADBIS 2018, LNCS, vol. 11019, pp. 263-277, Springer, Cham.

[11] Mancas, C. (2019) *MatBase* E-RD Cycles Associated Non-Relational Constraints Discovery Assistance Algorithm. In: Arai, K. et al. (eds.) Intelligent Computing. AISC 997, pp. 1-20, Springer Nature, Switzerland.

[12] Mancas, C. (2016) Algorithms for key discovery assistance. In: Repa, V., Bruckner, T. (eds). BIR 2016, LNBIP vol. 261, pp. 322-338, Springer, Cham.

[13] Mancas, C., Dorobantu, V. (2017) On enforcing relational constraints in *MatBase*. London Journal of Research in Comp. Sci. and Technology 17, 1 (Jan. 2017), 39-45.

[14] Mancas, C., Mocanu, A. (2017) *MatBase* DFS detecting and classifying E-RD cycles algorithm. Journal of Computer Science Applications and Information Technology 2 (4), 1–14.

[15] Melton, J., Simon, A.R. (2001) SQL 1999 Understanding Relational Language Components. Morgan Kaufmann, Burlington, MA.

[16] Murphy, S.V., Atala, A. (2014) 3D bioprinting of tissues and organs. Nature biotechnology 32, 773-785.

[17] Negus, C. (2015) Linux Bible. Wiley, Hoboken, NJ.

[18] Smith, J. (2018) Entity Framework Core in Action. Manning Publications Co., Shelter Island, NY.

[19] Thalheim, B. (2000) Entity-Relationship Modeling: Foundations of Database Technology. Springer-Verlag, Berlin.

AUTHORS

Christian Mancas has graduated in 1977 the Computers Department of the Politehnica University of Bucharest, Romania. He obtained his PhD degree in 1997 from the same as above Department. He first worked as a software engineer and, since 1980, R&D manager of a Computer Centre in Bucharest. Since 1990, he worked for several IT start-ups, including his owns, as data architect, software infrastructure manager, etc. Since 1998, he is an Associate Professor with both the Mathematics and Computer Science Department of the Ovidius University, Constanta and Engineering Taught in Foreign Languages Department (Computer Science and Telecommunications in English stream) of the Politehnica University, Bucharest, Romania (as an invited Professor). Christian Mancas published dozens of scientific papers, four books in Romanian, and one in English. He was a Program Committee member and session chairman for several software conferences in USA, Europe, Australia, and India. He is a member of several associations (including ACM, the Romanian Mathematics Sciences Society, and the International Who's Who of Professionals), and an editor of four U.S. soft-ware Journals. Since 2006, his biography is included in the Marquis' Who's Who in the World and Who's Who in Science and Technology, as well as in the Hubners' Who's Who in Romania. His main research areas are the conceptual data and knowledge modelling and querying, the db design, implementation, and optimization, as well as the architecture, design, development, fine-tuning, and maintenance of data and knowledge base management systems.