Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 2458-9403
Vol. 8 Issue 3, March - 2021

A Methodology for Refactoring Legacy DBs and DB Software Applications Using the (Elementary) Mathematical Data Model and *MatBase*

Christian Mancas
DATASIS ProSoft srl
Bucharest, Romania
christian.mancas@gmail.com

***Abstract*—This paper presents, in order, the steps needed to refactor any legacy db scheme and corresponding db software application by using the (Elementary) Mathematical Data Model and *MatBase* – the tool mainly based on it, but also on the Entity-Relationship and Relational Data Models. This methodology is fully illustrated with a case study: refactoring the MS Northwind Traders demo db. Both business analysis, architecture, design, and implementation flaws are tackled algorithmically, yielding both much higher quality refactored db schemes that guarantee data plausibility and the full set of the associated non-relational constraints needed to be enforced by the corresponding db software applications. The latter may be enforced by *MatBase* through automatically generated object-oriented and event-driven code.**

Keywords—database and database software application refactoring; the (Elementary) Mathematical Data Model; MatBase; conceptual data modeling; database design; non-relational constraints; MS Northwind Traders database

I. INTRODUCTION

Not only legacy databases (dbs) and database (db) software applications, but also currently developed ones are very often suffering from poor business analysis, architecture, design, and/or implementation.

While implementations are rather easily corrected or/and improved, for business analysis, architecture, and design powerful conceptual and technological tools are needed to both discover flaws and correct them.

We have since decades already introduced and used the (Elementary) Mathematical Data Model ((E)MDM) [13, 16] for the business analysis, architecture, and design of new dbs and corresponding db software applications. *MatBase* [13-17] is an intelligent prototype db and knowledge base management system based on both the Entity-Relationship (E-R) Data Model [7, 11, 20], the Relational one [1, 11], and, especially, the (E)MDM.

This paper proves that both the (E)MDM and *MatBase* may be successfully used as well for legacy ones, through a case study on the MS Northwind Traders demo db, provided since decades by Microsoft to all the users of both the MS Access and the MS SQL Server db management systems (DBMS) [18].

A. Related Work

Lot of research and practice accounting for db and db software application refactoring has been published. For example, the most recent ones are [2-5, 8, 9].

Similarly, lot of tools have also been developed for helping with such refactoring, e.g. [6, 10, 19].

Compared to all of them, both (E)MDM and *MatBase*, on one hand, provide everything that the above mentioned ones offer and, on the other, have many more powerful unique features, such as detecting and classifying E-R diagram (E-RD) cycles [17], assisting the business analysis process [11], detecting all existing key constraints [11, 12, 16], discovering of non-relational constraints associated to E-RD cycles [14, 16], guaranteeing the coherence and minimality of constraint sets [13, 16], etc.

B. Paper Outline

Section 2 briefly discusses the main flaws in the db architecture and design of the MS Northwind Traders demo db (for a detailed discussion, see [16]). Section 3 presents the corrected and enhanced relational db scheme obtained after refactoring it by using the (E)MDM and *MatBase*, as well as the associated set of non-relational constraints. The paper ends with conclusion, further work, and references.

II. MS NORTHWIND TRADERS MAIN FLAWS

The following 3 subsections briefly discuss the business analysis, general design, and table design main flaws detected in the MS Northwind Traders demo db. As it can be seen, this db scheme suffers from a lot of bad architectural and implementation decisions (for details, see Exercises 4.24 from [11], as well as 2.8 and 3.2 from [16]).

A. Business Analysis Main Flaws

For detecting the flaws of this type, the *MatBase* implementation of the Algorithm *A*0 (Assisting the Data Analysis and Modeling Process), introduced, discussed, and exemplified in [11], was applied on the corresponding E-R data model obtained in the Exercise 4.2 of [11]. Here are its main findings:

1. As they have exactly same structures, *Customers*, *Employees*, *Suppliers*, and *Shippers* seem to be several instances of a same object set. From their instances data, it is clear, however, that:
- *Customers* are people working for some companies.
- *Employees* are people working for the *Northwind Traders* company.
- *Suppliers* are people working for some supplying companies.
- *Shippers* are shipping companies (for which every employee data are always null).

Moreover, all four of them are heavily overloaded semantically, as neither companies, nor cities, states, countries, or job titles are text strings, but are object sets.

Consequently, a much better solution is to abstract instead two object sets, *People* and *Companies*, plus other four new ones, namely: *Job Titles*, *Cities*, *States/Provinces*, and *Countries/Regions* (as, in the MS solution, you can have, for example, some customers living in the city Seattle from the WA, U.S.A. state, and others from the same Seattle city, but from the Romanian state Dolj, which could be stored in their db as belonging to Brazil, etc.).

2. There is no need for 4 statuses object sets, which should be abstracted into only one. Moreover, the corresponding status name should be unique.

3. Both static sets *Categories* and *Payment Methods* should be replaced by dynamic ones: all the time new product categories and even payment methods (e.g., the bracelets used by the yearly UNTOLD music festival in Cluj-Napoca, Romania) are surfacing and it is not at all an easy task to modify code for each such new element in all corresponding db schemes and applications.

4. Not only for easing selection of the subset of employees from *People*, but especially for gaining generality and being able to sell this db application not only to *Northwind Traders*, but to any other interested company, an *Owner* object set is added for storing only the company that owns any copy of this db application.

5. For *Products*, there is a multivalued column *Supplier IDs*: while this MS Access recent feature may seem an advantage (and even "cool": unnormalization has lot of fans…), we strongly advise you not to use it ever: you cannot use SQL to search within such fields, or sort on them, or group by on them, or count their components, etc. For example, you cannot select with SQL all products offered by a supplier. Consequently, it was replaced with a regular *MainSupplier* mapping (column) and a *Supplying Options* (relationship type) object set, for storing any additional suppliers, product by product, if any.

6. Finally, the worse db design flaw, which renders this demo db unusable by any trade company is embedded in the text mapping (column) *Quantity Per Unit* also of the set (table) *Products*: its values are, for example, "10 boxes x 20 packs", "12 bottles of 550 ml", "36 boxes", etc. Trivially, for searching within them you need parsing and semantical analysis… No wonder that there is no total amount for customer orders, that the *Amount Due* in *Invoices* must be manually established, no unitary price comparisons are possible, no automatic stock maintenance is possible, etc., etc., etc. Consequently, this mapping (column) was modified to take positive natural values and new sets (tables) for *MeasureUnits* and *PackTypes* were added as codomains for newly added mappings (columns) *MeasureUnit*, *Package*, and *Wrapping*, respectively; also added were corresponding natural mappings (columns) *PackageNo* and *WrappingNo*.

B. General Design Flaws

Next, the *MatBase* implementation of Algorithm *A*1 (Translating E-R data models into (E)MDM schemes) [16] was applied for automatically obtaining the corresponding initial (E)MDM scheme. Then, applying the *MatBase* implementation of Algorithm *A*2 (Assisting validation and enhancement of initial (E)MDM schemes) [16] yielded the following general and particular set (table) and function (column) types of design flaws:

-A few codomain (range) constraints have been added (i.e. phone numbers are naturals of at most 12 digits) and others have been modified (e.g., UNICODE(50) was replaced by UNICODE(64), with 64 being the smallest power of 2 greater than 50).

-Nearly all number mappings (columns) have been restricted to their positive subsets (e.g., there may not be negative quantities, costs, prices, a.s.o.).

-Many totality (NOT NULL) constraints were added (e.g., it is impossible not to know (from) where people or companies work, at least a main supplier for any product, their quantities per unit, neither the first, nor the last names of employees and customers, etc.).

-Another very severe general design flaw is the almost complete lack of semantic keys (see, for

example, the best practice rule R-K-2 from subsection 3.11.1.2 of [11]). It is not acceptable to distinguish between elements of any object set only by their autonumbers. Consequently, all obvious key constraints have been added too (and documented, see next subsection) in this step.

-For all mappings (columns) that cannot be used for unique identification of their domain sets (tables), corresponding nonprime constraints were added as well.

-All tuple (check) constraints shown in subsection C.1 were discovered and added as well.

-As the structural functions (foreign keys) *Customer Order* and *Purchase Order* of the *Inventory Transactions* should be automatically generated, they have been turned into corresponding computed mappings (columns).

-As the inventory transaction foreign keys should be automatically generated too, both *Inventory* structural functions became computed as well. As such, their one-to-oneness need not be enforced (but at least during the db software application testing process this could be useful).

-It is better to add autonumber primary key *ID*s to both *Employee Privileges* and *Supplying Options*, so that future extensions of this db scheme may reference them naturally and faster.

-*Group By* of *Sales Reports* should not be unique (as there might be several sales reports grouped by a same column).

-There is no need of the Boolean mapping (column) *Posted to Inventory*, as its values may be computed from the corresponding **Inventory* one.

-For homogeneity, all primary keys have been abbreviated as *ID*.

-All structural functions have been renamed by removing their suffix *ID* (e.g., *Order ID* of *Invoices* is renamed as *Order*). See best practice rule R-DA-1 from subsection 2.10.1 of [11].

Next, the *MatBase* implementation of Algorithm *A*4.0 (Detecting and classifying E-RD cycles) [16, 17] was run on this corrected, enhanced, and validated (E)MDM scheme, yielding a total of 75 cycles. They were next analyzed with the help of the *MatBase* implementation of Algorithm *A*4 (Assistance of the E-RD cycles analysis) [14, 16], which helped discovery of the constraints that are presented in the subsection 3.B, which were also added to the enhanced db scheme.

C. Table by Table Main Flaws

Next, the *MatBase* implementation of Algorithm *A*3" (Assistance of detecting keys) [11, 12, 16] was run for each set (table), yielding the following semantic key constraints (where • denotes concatenation of columns; mathematically: the Cartesian function product operator):

1. For *Doc. Statuses*: *Status Name* (There should not be two statuses having same names.)
2. For *Categories*: *Category* (There should not be two categories having same names.)
3. For *PaymMethods*: *PayMethod* (There may not be two payment methods having same names.)
4. For *Privileges*: *Privilege Name* (There should not be two privileges having same names.)
5. For *Countries/Regions*: *Country/Region* (There may not be two countries / regions having same names.)
6. For *States/Provinces*: *Country/Region* • *State/Province* (There may not be two states/provinces of a same country/region having same names.)
7. For *Cities*: *City* • *Zip/Postal Code* • *State/Province* (There may not be two cities of a same state/province having same names and zip / postal codes.)
8. For *Job Titles*: *Job Title* (There should not be two job titles having same names.)
9. For *MeasureUnits*: *MeasureUnit* (There may not be two measure units having same symbols.)
10. For *PackTypes*: *PackType* (There should not be two package types having same names.)
11. For *People*:

– *E-mail Address* (There should not be two persons using a same e-mail address.)

– *Mobile Phone* (There should not be two persons using a same mobile phone.)

– *Web Page* (There should not be two persons using a same web page.)

– *First Name* • *Last Name* • *Business Phone* (There should not be two persons having same first and last names, and using a same business phone, as it would confuse both colleagues, customers, and shippers.)

– *First Name* • *Last Name* • *Company* • *Job Title* (There should not be two persons having same first and last names, and employed by a same company in same jobs, as it would confuse both colleagues, customers, and shippers.)

12. For *Companies*:

– *E-mail Address* (There may not be two companies using a same e-mail address.)

– *Business Phone* (There may not be two companies using a same business phone.)

– *Mobile Phone* (There may not be two companies using a same mobile phone.)

– *Fax Number* (There should not be two companies using a same fax line.)

– *Web Page* (There may not be two companies using a same web page.)

– *Company* • *City* • *Address* (There may not be two companies having same name and street address in a same city.)

– *Company* • *City* • *Zip/Postal Code* (There may not be two companies having same names and zip / postal codes in a same city.)

13. For *Products*:

– *Product Code* (There should not be two products having same code.)

– *Product Name* • *Category* (There should not be two products of a same category having same names.)

14. For *Orders*: *Order Date* • *Employee* • *Customer* (There should not be two orders established in a same day, by a same employee, for a same customer.)

15. For *Order Details*:

– **Inventory* (There should not be two inventory transactions generated by the same order detail. Automatically generated by the application when posting corresponding order detail to the *Inventory Transactions*.)

– *Order* • *Product* (There should not be two order details of a same order for a same product.)

16. For *Invoices*: *Order* • *Invoice Date* (There should not be two invoices for a same order issued in a same day.)

17. For *Purchase Orders*: *Creation Date* • *Supplier* • *Created By* (There should not be two purchase orders created in a same day, by a same employee, for a same supplier.)

18. For *Purchase Order Details*:

– **Inventory* (There should not be two inventory transactions generated by the same purchase order detail. Automatically generated by the application when posting corresponding purchase order detail to the *Inventory Transactions*.)

– *Purchase Order* • *Product* (There should not be two purchase order details of a same purchase order for a same product.)

19. For *Inventory Transaction Types*: *Type Name* (There should not be two inventory transaction types having same names.)

20. For *Inventory Transactions*:

– *Product* • **Customer Order* (For each inventory transaction corresponding to a detail customer order there is only one such detail row for any product.)

– *Product* • **Purchase Order* (For each inventory transaction corresponding to a detail purchase order there is only one such detail row for any product.)

21. For *Employee Privileges*: *Employee* • *Privilege* (It would be senseless to store twice that an employee has a right.)

22. For *Supplying Options*: *Supplier* • *Product* (It would be senseless to store twice that a company supplies a product.)

23. For *Sales Reports*: *Title* (There should not be two reports having same names.)

24. For *Strings*: *String Data* (There is no use in storing a same string twice.)

In the next step, the *MatBase* implementation of Algorithms *A*5 (Assistance for guaranteeing the coherence of constraint sets) and *A*6 (Guaranteeing minimality of constraint sets) [13, 16] were run on this db scheme without changing it, as it is both coherent and minimal.

III. CORRECTED AND ENHANCED SOLUTION

The *MatBase* implementation of Algorithm *A*7 (Automatic translation of (E)MDM schemes into relational ones and associated sets of non-relational constraints) [16] was run on the above final (E)MDM scheme. Its output is presented in the next two subsections. In the process, *MatBase* automatically generated both the forms of the refactored corresponding db software application and the object-oriented event-driven code needed to enforce the non-relational constraints [15, 16].

For the time being, queries and reports from the legacy db application must be manually refactored, if needed. The same goes for populating the refactored db with the corresponding legacy one instance.

A. The Relational DB Schema and a Plausible DB Instance

The conventions used for relational db (rdb) schemes are those described in [11]. Here, we remind only the needed ones for understanding what follows, namely:

- The parentheses after table names contain their keys (with the primary ones underlined).
- Between these parentheses and the tables, tuple (check) constraints are written (if any).
- The first header lines of the tables contain the column names (in italics, with computed columns prefixed by *).
- The second header lines contain domain and referential integrity (foreign key) constraints. *Im*(*T*) is an abbreviation for *Im*(*T.ID*), where *Im*(*f*) denotes the image of mapping *f* (i.e., the set of values taken by *f*). Consequently, for any foreign key column *fk*, the associated referential integrity constraint *Im*(*T*) states that *fk* references the primary key (*ID*) of the table *T*.
- The third header lines are reserved for the NOT NULL constraints.
- AUTON(*n*) denotes the set of autonumbers (i.e., the unique integers automatically generated by the system) with at most *n* digits.
- NAT(*n*), INT(*n*), CURRENCY$^+$(*n*) denote the subsets of, respectively, naturals, integers, and positive currencies with at most *n* digits.
- RAT$^+$(*n*, *m*) denotes the subset of the positive rational (floating point) numbers with at most *n* digits, out of which the last *m* are the decimal ones.
- UNICODE(*n*) denotes the subset of strings of length at most *n* made of UNICODE characters.

Owner

OwnerCompany
Im(*Companies*)
NOT NULL
0

Inventory Transaction Types (*ID*, *Type Name*)

ID	*Type Name*
AUTON(2)	UNICODE(32)
NOT NULL	NOT NULL
1	Bought
2	Sold

Countries / Regions (*ID*, *Country / Region*)

ID	*Country / Region*
AUTON(3)	UNICODE(64)
NOT NULL	NOT NULL
1	U.S.A.
2	Romania

PaymMethods (*ID*, *PayMethod*)

ID	*PayMethod*
AUTON(1)	UNICODE(16)
NOT NULL	NOT NULL
1	Card
2	Cash

Privileges (*ID*, *Privilege Name*)

ID	*Privilege Name*
AUTON(2)	UNICODE(32)
NOT NULL	NOT NULL
1	Purchase Approvals

Strings (*ID*, *String Data*)

ID	*String Data*
AUTON(9)	UNICODE(255)
NOT NULL	NOT NULL
1	Northwind Traders
2	Continue?

Supplying Options (*ID*, *Supplier* • *Product*)

ID	*Supplier*	*Product*
AUTON(12)	*Im*(*Suppliers*)	*Im*(*Products*)
NOT NULL	NOT NULL	NOT NULL
1	2	1
2	2	2

Employee Privileges (*ID*, *Employee* • *Privilege*)

ID	*Employee*	*Privilege*
AUTON(12)	*Im*(*People*)	*Im*(*Privileges*)
NOT NULL	NOT NULL	NOT NULL
1	0	1
2	1	1

Job Titles (*ID*, *Job Title*)

ID	*Job Title*
AUTON(2)	UNICODE(32)
NOT NULL	NOT NULL
1	Salesman
2	VP Sales
3	Sales Manager

MeasureUnits (*ID*, *MeasureUnit*)

ID	*MeasureUnit*
AUTON(2)	UNICODE(16)
NOT NULL	NOT NULL
0	kg
1	g
2	ml

Doc. Statuses (*ID*, *Status Name*)

ID	*Status Name*
AUTON(2)	UNICODE(32)
NOT NULL	NOT NULL
1	New
2	No stock
3	Taxable

PackTypes (*ID*, *PackType*)

ID	*PackType*
AUTON(2)	UNICODE(16)
NOT NULL	NOT NULL
0	box
1	bag
2	bottle

Categories (*ID*, *Category*)

ID	*Category*
AUTON(3)	UNICODE(32)
NOT NULL	NOT NULL
1	Beverages
2	Spices
3	Pasta

States / Provinces (*ID*, *Country / Region* • *State / Province*)

ID	*Country / Region*	*State / Province*
AUTON(5)	*Im*(*Countries / Regions*)	UNICODE(64)
NOT NULL	NOT NULL	NOT NULL
1	1	WA
2	2	B

Cities (*ID*, *City* • *Zip / Postal Code* • *State / Province*)

ID	*City*	*State / Province*	*Zip / Postal Code*
AUTON(7)	UNICODE(64)	*Im*(*States / Provinces*)	UNICODE(16)
NOT NULL	NOT NULL	NOT NULL	NOT NULL
1	Seattle	1	98000
2	Bucharest	2	700000

Companies (*ID*, *E-mail Address*, *Fax Number*, *Business Phone*, *Mobile Phone*, *Web Page*, *Company* • *City* • *Address*, *Company* • *City* • *Zip / Postal Code*)

ID	*Company*	*City*	*Zip / Postal Code*
AUTON(9)	UNICODE(64)	*Im*(*Cities*)	UNICODE(16)
NOT NULL	NOT NULL	NOT NULL	NOT NULL
0	Northwind Traders	1	98101
1	Supplier A	1	98101
2	Company AA	1	98191

Address	*E-mail Address*	*Business Phone*	*Mobile Phone*
UNICODE(4096)	UNICODE(64)	NAT(12)	NAT(12)
NOT NULL	NOT NULL	NOT NULL	NOT NULL
1, 1st Avenue S	off@northwindtrad.com	1236660000	3216660000
1, Olive Way	SupplA@gmail.com	1235550100	3215550100
2, Pine Str.	CompAA@gmail.com	1237770200	3217770200

Fax Number	*Web Page*	*Notes*	*Attachements*
NAT(12)	HYPERLINK	UNICODE(4096)	ATTACHMENT

People (*ID*, *E-mail Address*, *Mobile Phone*, *Web Page*, *First Name* • *Last Name* • *Business Phone*, *First Name* • *Last Name* • *Company* • *Job Title*)

ID	*First Name*	*Last Name*	*Zip / Postal Code*
AUTON(12)	UNICODE(64)	UNICODE(64)	UNICODE(16)
NOT NULL	NOT NULL	NOT NULL	NOT NULL
0	Andrew	Cencini	98101
1	Laura	Giussani	98161

Address	*E-mail Address*	*Business Phone*	*Mobile Phone*
UNICODE(4096)	UNICODE(64)	NAT(12)	NAT(12)
NOT NULL			
123, 2nd Avenue	andrew@northwindtraders.com	1235550100	1235550102
123, 8th Avenue	laura@northwindtraders.com	1235550100	1235550103

City	*Company*	*Job Title*
Im(*Cities*)	*Im*(*Companies*)	*Im*(*Job Titles*)
NOT NULL		
1	0	2
1	0	3

Fax Number	*Web Page*	*Notes*	*Attachements*

NAT(12)	HYPERLINK	UNICODE(4096)	ATTACHMENT

Products (*ID*, *Product Code*, *Product Name* • *Category*)

ID	*Product Name*	*Category*	*Product Code*
AUTON(9)	UNICODE(64)	*Im*(*Categories*)	UNICODE(25)
NOT NULL	NOT NULL	NOT NULL	NOT NULL
1	Northwind Traders tea	1	NWTB-1
2	Northwind Traders syrup	2	NWTCO-3

Package	*PackageNo*	*Wrapping*	*WrappingNo*
Im(*PackTypes*)	NAT(2)	*Im*(*PackTypes*)	NAT(2)
NOT NULL	NOT NULL		
1	20	0	10
2	12		

MeasureUnit	*Quantity Per Unit*	*Standard Cost*	*List Price*
Im(*MeasureUnits*)	$RAT^{+}(18,4)$	$CURRENCY^{+}(14)$	$CURRENCY^{+}(14)$
NOT NULL	NOT NULL	NOT NULL	NOT NULL
2	10	$13.50	$18.00
3	550	$7.50	$10.00

MainSupplier	*Minimum Reorder Qty*	*Reorder Level*	*Target Level*
Im(*Companies*)	NAT(9)	NAT(9)	NAT(9)
NOT NULL			
1	10	10	40
1	25	25	100

Description	*Discontinued*	*Attachments*
UNICODE(4096)	BOOLEAN	ATTACHMENT

Sales Reports (*ID*, *Title*)

ID	*Title*	*Display*	*Group By*
AUTON(2)	UNICODE(64)	UNICODE(32)	UNICODE(32)
NOT NULL	NOT NULL	NOT NULL	NOT NULL
1	Sales by Product	Product	Product
2	Sales by Category	Category	Category

Default	*Filter Row Source*
BOOLEAN	UNICODE(4096)
	NOT NULL
✓	SELECT DISTINCT [Product Name] FROM [Products] ORDER BY [Product Name];
	SELECT DISTINCT [Category] FROM [Categories] ORDER BY [Category];

Orders (*ID*, *Order Date* • *Customer* • *Employee*) *Status* ≠ *Tax Status* ∧ *Shipped Date* ≥ *Order Date* ∧ *Paid Date* ≥ *Order Date*

ID	*Order Date*	*Customer*	*Employee*
AUTON(9)	[1/1/1900, 12/31/9999], *now*()	*Im*(*People*)	*Im*(*People*)
NOT NULL	NOT NULL	NOT NULL	NOT NULL
1	3/6/2006	1	1
2	4/3/2006	0	0

Shipped Date	*Paid Date*	**TotAmount*	*Ship Customer*	*Shipper*
[1/1/1900, 12/31/9999]	[1/1/1900, 12/31/9999]	$CURRENCY^{+}(14)$, 0	*Im*(*People*)	*Im*(*Companies*)
				NOT NULL

3/9/2006	3/6/2006	$48		2
4/3/2006	4/3/2006	$10		2

Shipping Fee	*Taxes*
CURRENCY$^+$(14), 0	CURRENCY$^+$(14), 0
NOT NULL	NOT NULL
$12.00	$0.00
$0.00	$0.00

Notes	*Tax Rate*	*Ship Address*	*Ship ZIP / Postal Code*
UNICODE(4096)	NAT(2), 0	UNICODE(4096)	UNICODE(16)
	NOT NULL		
	0%		
	0%		

Ship City	*Payment Type*	*Status*	*Tax Status*
Im(*Cities*)	*Im*(*PaymMethods*)	*Im*(*Doc. Statuses*), 0	*Im*(*Doc. Statuses*)
		NOT NULL	
	1	0	
	1	0	

Order Details (*ID*, **Inventory*, *Order* • *Product*)

ID	*Order*	*Product*	*Quantity*
AUTON(10)	*Im*(*Orders*)	*Im*(*Products*)	RAT$^+$(18,4)
NOT NULL	NOT NULL	NOT NULL	NOT NULL
1	2	1	2
2	2	2	1

Discount	*Unit Price*	*Status*
[0, 1], 0	CURRENCY$^+$(14)	*Im*(*Doc. Statuses*)
NOT NULL	NOT NULL	NOT NULL
0.00%	$18.00	1
0.00%	$10.00	1

Date Allocated	**Inventory*	*Purchase Order*
[1/1/1900, 12/31/9999]	*Im*(*Inventory Transactions*)	*Im*(*Purchase Orders*)
	NOT NULL	
	1	
	2	

Invoices (*ID*, *Order* • *Invoice Date*) *Due Date* > *Invoice Date*

ID	*Order*	*Invoice Date*	*Due Date*
AUTON(9)	*Im*(*Orders*)	[1/1/1900, 12/31/9999], *now*()	[1/1/1900, 12/31/9999]
NOT NULL	NOT NULL	NOT NULL	NOT NULL
1	2	3/9/2006	4/9/2006
2	2	4/3/2006	5/3/2006

Amount Due	*Shipping*	*Tax*
CURRENCY$^+$(14)	CURRENCY$^+$(14), 0	CURRENCY$^+$(14), 0
NOT NULL	NOT NULL	NOT NULL
$0.00	$0.00	$0.00
$0.00	$0.00	$0.00

Purchase Orders (*ID*, *Creation Date* • *Supplier* • *Created By*) *Creation Date* ≤ *Submitted Date* ≤ *Approved Date* ≤ *Payment Date* ≤ *Expected Date*

ID	*Creation Date*	*Supplier*	*Created By*
AUTON(9)	[1/1/1900, 12/31/9999], *now*()	*Im*(*Companies*)	*Im*(*People*)

NOT NULL	NOT NULL	NOT NULL	NOT NULL
1	3/6/2006	2	1
2	4/3/2006	2	0

Submitted Date	*Approved Date*	*Payment Date*	*Payment Amount*	*Submitted By*
[1/1/1900, 12/31/9999]	[1/1/1900, 12/31/9999]	[1/1/1900, 12/31/9999]	CURRENCY^{+}(14), 0	*Im*(*People*)
3/6/2006	3/9/2006	3/9/2006	$40	1
4/3/2006	4/3/2006	4/3/2006	$8	0

Shipping Fee	*Taxes*	*Approved By*	*Notes*
CURRENCY^{+}(14), 0	CURRENCY^{+}(14), 0	*Im*(*People*)	UNICODE(4096)
NOT NULL	NOT NULL		
$12.00	$0.00	0	
$0.00	$0.00	0	

Payment Method	*Expected Date*	*Status*
Im(*PaymMethods*)	[1/1/1900, 12/31/9999]	*Im*(*Doc. Statuses*), 0
		NOT NULL
1	3/23/2006	0
1	4/17/2006	0

Purchase Order Details (*ID*, **Inventory*, *Purchase Order • Product*)

ID	*Purchase Order*	*Product*	*Quantity*
AUTON(10)	*Im*(*Purchase Orders*)	*Im*(*Products*)	RAT^{+}(18,4)
NOT NULL	NOT NULL	NOT NULL	NOT NULL
1	1	1	2
2	1	2	1

Unit Cost	*Date Received*	**Inventory*
CURRENCY^{+}(14)	[1/1/1900, 12/31/9999]	*Im*(*Inventory Transactions*)
NOT NULL		
$14.00	3/20/2006	3
$8.00	4/10/2006	4

Inventory Transactions (*ID*, **Customer Order • Product*, **Purchase Order • Product*)
Transaction Modified Date ≥ Transaction Created Date

ID	*Transaction Created Date*	**Customer Order*	**Purchase Order*	*Product*
AUTON(9)	[1/1/1900, 12/31/9999], *now*()	= *Order* ∘ **Inventory*$^{-1}$	= *Purchase Order* ∘ **Inventory*$^{-1}$	*Im*(*Products*)
NOT NULL	NOT NULL			NOT NULL
1	3/6/2006	1		1
2	3/6/2006	2		2
3	4/3/2006		1	1
4	4/3/2006		2	2

Transaction Modified Date	*Quantity*	*Transaction Type*	*Comments*
[1/1/1900, 12/31/9999], *now*()	RAT^{+}(18,4)	*Im*(*Inventory Transaction Types*)	UNICODE(255)
	NOT NULL	NOT NULL	
	2	2	
	1	2	
	2	1	
	1	1	

MeasureUnits (*ID*, *MeasureUnit*)

ID	*MeasureUnit*	*MultipleOf*	*MultiplicityOrder*
AUTON(2)	UNICODE(16)	*Im*(*MeasureUnits*)	$NAT^{+}(3)$
NOT NULL	NOT NULL		
0	g		
1	kg	0	1000
2	ml		

B. The Associated Non-Relational Constraint Set

The following non-relational constraints associated to the above presented rdb scheme were enforced by *MatBase* in the refactored db software application, through automatically generated forms and object-oriented event-driven code [15, 16]:

OC: *card*(*Owner*) = 1 (Any db application copy may be owned by only one company.)

PBPC: (∀*x*,*y*∈*People*)(*Business Phone*(*x*) = *Business Phone*(*y*) ⇒ *Company*(*x*) = *Company*(*y*)) (Any two employees using a same business phone are working for a same company.)

SOC: (∀*x*∈*Supplying Options*)(*Supplier*(*x*) ≠ *MainSupplier*(*Product*(*x*))) (Whenever the fact that a supplier is supplying a product is stored by *MainSupplier*, it should not be also stored by *Supplying Options*.)

OEC: *Ship Customer* ⊢ *Ship City* • *Ship Address* • *Ship ZIP/Postal Code* (Whenever ship customer is specified, both ship city, address, and zip / postal code must be specified too.)

OOEC: (∀*x*∈*Orders*)(*Company*(*Employee*(*x*)) = *OwnerCompany*(1)) (Any employee must be employed by the owner company.)

ODDC: (∀*x*∈*Order Details*)(*Order Date*(*Order*(*x*)) ≤ *Date Allocated*(*x*) ≤ *Shipped Date*(*Order*(*x*))) (*Date Allocated* should be between the corresponding orders' *Order Date* and *Shipped Date*.)

ODACC: (∀*x*∈*Order Details*)(*Status*(*x*) ≠ *Tax Status*(*Order*(*x*))) (The customer orders' tax statuses may not be the same as their statuses.)

ODCC: (∀*x* ∈ *Order Details*)(∀*y* ∈ *Inventory Transactions*)(*Order*(*x*) = *Customer Order*(*y*) ⇒ *Product*(*x*) = *Product*(*y*) ∧ *Quantity*(*x*) = *Quantity*(*y*) ∧ *Date Allocated*(*x*) ≤ *Transaction Created Date*(*x*)) (The product and quantity of any inventory transaction corresponding to a customer order detail should be the same as the one of that customer order detail and the transaction created date may be at least equal to the corresponding allocating date.)

ODPACC: (∀*x*∈*Order Details*)(*Status*(*Purchase Order*(*x*)) ≠ *Tax Status*(*Order*(*x*))) (The purchase orders' tax statuses may not be the same as any of their corresponding customer orders' statuses.)

ODPSACC: (∀*x*∈*Order Details*)(*Status*(*Purchase Order*(*x*)) ≠ *Status*(*x*)) (Order details' statuses are always distinct of the statuses of their corresponding purchase orders.)

IADC: (∀*x*∈*Invoices*)(*Amount Due*(*x*) ≤ **TotAmount*(*Order*(*x*))) (For any invoice, its amount due value may not be greater than the total one of the corresponding order.)

EPC: (∀*x*∈*Employee Privileges*)(*Company*(*Employee*(*x*)) = *OwnerCompany*(1)) (Only employees of the owner company may have privileges on the db application objects.)

POEC: (∀*x*∈*Purchase Orders*)(*Company*(*Created By*(*x*)) = *Company*(*Submitted By*(*x*)) = *Company*(*Approved By*(*x*)) = *OwnerCompany*(1)) (All persons that are creating, submitting, and approving purchase orders must be employed by the owner company.)

PODDC: (∀*x*∈*Purchase Order Details*)(*Approved Date*(*Purchase Order*(*x*)) ≤ *Date Received*(*x*) ≤ *Payment Date*(*Purchase Order*(*x*))) (*Date Received* should be between the corresponding purchase orders' *Approved Date* and *Payment Date*.)

PODPCC: (∀*x*∈ *Purchase Order Details*)(∀*y* ∈ *Inventory Transactions*)(*Purchase Order*(*x*) = *Purchase Order*(*y*) ⇒ *Product*(*x*) = *Product*(*y*) ∧ *Quantity*(*x*) = *Quantity*(*y*) ∧ *Date Received*(*x*) ≤ *Transaction Created Date*(*x*)) (The product and quantity of any inventory transaction corresponding to a purchase order detail should be the same as the one of that purchase order detail and the transaction created date may be at least equal to the corresponding receiving date.)

*GCC*1: (∀*x* ∈ *Purchase Order Details*)(∀*y* ∈ *Order Details*)(*Purchase Order*(*x*) = *Purchase Order*(*y*) ⇒ *Product*(*x*) = *Product*(*y*)) (Whenever an order detail has an associated purchase order, that purchase order should have a detail one for the same product as the one of the order detail.)

*GCC*2: (∀*x* ∈ *Purchase Order Details*)(∀*y* ∈ *Order Details*)(*Purchase Order*(*Inventory*(*x*)) = *Purchase Order*(*y*) ⇒ *Product*(*x*) = *Product*(*y*)) (Whenever an order detail has an associated purchase order, that purchase order should have a detail one for the same product as the one of the order detail and to which it corresponds an inventory transaction for that purchase order.)

NEC: ¬ (*Customer Order*, *Purchase Order*) (There must not be inventory transactions for which both customer and purchase orders are specified.)

PODRDC: (∀*x* ∈ *Purchase Order Details*)(*Date Received*(*x*) ≥ *Payment Date*(*Purchase Order*(*x*))) (Purchase ordered products' receiving dates may not be prior to corresponding purchase orders' payment dates; of course that, whenever special arrangements were made with suppliers such that shipments may arrive before corresponding payments, *Payment Date* should be replaced by *Approved Date*.)

*ITTC*1: (∀*x*∈*Inventory Transactions*)(*Customer Order*(*x*) ∉ NULLS ⇒ *Transaction Type*(*x*) ∈ {2, 3}) (Inventory transactions corresponding to customer orders may only be of types "Sold" and "Reserved".)

*ITTC*2: (∀*x*∈*Inventory Transactions*)(*Purchase Order*(*x*) ∉ NULLS ⇒ *Transaction Type*(*x*) = 1) (Inventory transactions corresponding to purchase orders may only be of type "Bought".)

*ITTC*3: (∀*x*∈*Inventory Transactions*)(*Purchase Order*(*x*) ∈ NULLS ∧ *Customer Order*(*x*) ∈ NULLS ⇔ *Transaction Type*(*x*) = 4) (Inventory transactions not corresponding to either customer or purchase orders may only be of type "Waste".)

IV. CONCLUSION AND FURTHER WORK

We have presented, in order, the steps needed to refactor any legacy db scheme by using the (E)MDM and *MatBase*. We fully illustrated this methodology with a case study: refactoring the MS Northwind Traders demo db scheme and software application.

Due to paper length limitations, only main flaws and step results are presented; for full details, see Exercises 2.8 and 3.2 from [16]. Both business analysis, architecture, design, and implementation flaws were tackled algorithmically.

We obtained both a much higher quality refactored db scheme that guarantees data plausibility and the full set of the associated non-relational constraints that were enforced in the corresponding db software refactored application through *MatBase* automatically generated object-oriented and event-driven code.

The 12 steps of this original proposed methodology, solely based on the (E)MDM algorithms implemented in *MatBase* (some of them providing only assistance to the db designers, whenever the steps or some of their sub-steps cannot be automatically performed), are the following:

1. Reverse engineer the legacy db scheme into an E-R data model (automatically).
2. Redo the business analysis to correct and enhance the E-R data model (assisted).
3. Translate the refactored E-R data model into an initial (E)MDM scheme (automatically).
4. Analyze, correct, and enhance the initial scheme (assisted).
5. Detect and classify all cycles in the corresponding E-RD (automatically).
6. Analyze these cycles and detected all associated non-relational constraints (assisted).
7. Detect all keys for all sets/tables (assisted).
8. Check and enforce the constraint set coherence (assisted).
9. Enforce the constraint set minimality (automatically).
10. Translate the final (E)MDM scheme above obtained into the corresponding refactored rdb scheme, its associated non-relational constraint set (automatically), and refactor the corresponding db management software application (assisted).
11. Enforce the non-relational constraints through automatically generated code (automatically).
12. Populate the refactored rdb with the instance of the legacy one (assisted).

Further work will be done to automate as much as possible the processes of refactoring the queries and reports of the legacy db applications (i.e., last substep of step 10) and the one of extracting data instances from the legacy dbs and populating the corresponding refactored ones with it (i.e., step 12).

As seen from both this paper and [11, 16], the MS Northwind Traders demo db is, in fact, rather a counterexample of a db scheme and software application, which is not at all guaranteeing data plausibility (as it does not enforce at least all existing basic constraints as not nulls, domain, and key ones). Similarly, as it does not enforce the associated existing non-relational constraints, the corresponding db software application is not helping users to maintain their data quality. Moreover, through an incredible bad design decision for a column, it is not at all useful for any trading company.

Unfortunately, this is not at all an exception: from the similar MS SQL Server Pubs, Oracle Express HR, MySQL Sakila, MySQL World, and DB2 Express-C Sample dbs to most of the hundreds of thousand commercial legacy dbs, all of them would greatly benefit from being refactored with the methodology proposed in this paper.

Fortunately, however, some of these missing constraints are enforced by its associated db software application. Nonetheless, as already discussed in [11], this is not at all ideal, from several points of view, out of which the more important are the following ones:

-software applications are always prone to errors;

-rather often, DBAs are working directly with dbs, bypassing the software applications (generally for trying to recover updates made between the last available backups and system/application crashes), which may result in implausible db instances;

-lack of algorithmic business analysis and db design favors non-discovery of all existing business rules, especially in complex subuniverses, which then allows storing implausible data.

Consequently, it is always much better to discover all business rules in any subuniverse of discourse, formalize them, and then enforce them through DBMSes (for the relational ones provided by the chosen DBMS versions) and corresponding db software applications (for both the non-relational ones and the relational ones that are not provided by the chosen DBMS versions).

REFERENCES

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

[2] Ambler, S. W., Sadalage, P. J. (2006). *Refactoring Databases: Evolutionary Database Design*. Addison-Wesley Professional.

[3] Ambysoft Inc. (2020). The Process of Database Refactoring: Strategies for Improving Database Quality. http://agiledata.org/essays/databaseRefactoring.html

[4] Ambysoft Inc. (2020). Catalog of Database Refactorings.http://www.agiledata.org/essays/databaseRefactoringCatalog.html

[5] Birchall, C. (2016). *Re-Engineering Legacy Software*. Manning Publications.

[6] Blu Age (2020). Blue Age Database and data Modernization. https://www.bluage.com/products/db-modernization

[7] Chen, P.P. (1976). The entity-relationship model: Toward a unified view of data. ACM TODS 1(1): 9-36.

[8] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

[9] Harrison, N. (2011). A Developers' Guide to Refactoring Databases. https://www.red-gate.com/simple-talk/opinion/opinion-pieces/a-developers-guide-to-refactoring-databases/

[10] Leiloff, L. et al. (2017). *DELTA – A Tool for Database Refactoring.* https://www.researchgate.net/publication/320986991_DELTA_-_A_tool_for_database_refactoring

[11] 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), Palm Bay, FL.

[12] Mancas, C. (2016a). Algorithms for Key Discovery Assistance. In: Repa, V., Bruckner, T. (eds). BIR 2016, Lecture Notes in Business Information Processing vol. 261, pp. 322–338. Springer, Cham, Switzerland.

[13] Mancas, C. (2018). *MatBase* Constraint Sets Coherence and Minimality Enforcement Algorithms. In: Benczur, A., Thalheim, B., Horvath, T. (eds.), Proc. 22nd ADBIS Conf. on Advances in DB and Inf. Syst., LNCS 11019, pp. 263–277. Springer, Cham, Switzerland.

[14] Mancas, C. (2019). *MatBase* E-RD Cycles Associated Non-Relational Constraints Discovery Assistance Algorithm. In: Arai, K., Bhatia, R., Kapoor, S. (eds.), *Intelligent Computing*, Proc. 2019 Computing Conference, AISC Series 997.1 (2019):390–409. Springer, Cham, Switzerland.

[15] Mancas, C. (2019). *MatBase* - a Tool for Transparent Programming while Modeling Data at Conceptual Levels. In: Proc. 5th Int. Conf. on Comp. Sci. & Inf. Techn. (CSITEC 2019), pp. 15–27. AIRCC Pub. Corp., Chennai, India.

[16] Mancas, C. (2021). *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), Palm Bay, FL (in press).

[17] Mancas, C., Mocanu, A. (2017). *MatBase* DFS Detecting and Classifying E-RD Cycles Algorithm. J. Comp. Sci. App. and Inform. Techn. 2(4):1–14.

[18] Microsoft Corp. (2016). *Northwind Database.* https://access-templates.com/tag/northwind.html

[19] Quest Software Inc. (2020). *ApexSQL Refactor.* https://solutioncenter.apexsql.com/sql-database-refactoring-introduction/

[20] Thalheim, B. (2000). *Entity-Relationship Modeling. Foundations of Database Technology*. Springer-Verlag.