Application of Relational Normalforms in CASE-tools

Béla Halassy

Computing Application and Service Co. (SZAMALK)
Szakasits A. u. 68., 1502 Budapest, Hungary

Abstract

A new interpretation of relational normalforms is discussed. The goal is to present such a normalization algorithm that aids process-design, too. Practical experiences of applying normalform synthesis and decomposition are highlighted. NF problems are recited and reinterpreted. Normalforms are revised for completeness. The notion of semantical normalization is explained. The concept of 'thread' versus 'cover' is suggested. The use of threads in a coupled data- and process-design is recommended.

1. Background

Based on a long (1976-1987) theoretical and practical experimentation, our research group has prepared a CASE-tool, named SYDES. This acronym stands for SYstems DEsigner System. The name covers an information systems design method, a software supporting the method and a systems theory, which is the frame of both.

SYDES is a Ianus-faced product. It is a CASE-shell, by the aid of which arbitrary categories (object-types) can be defined. Analysts may specify design attributes of categories, conventional values of attributes and informal descriptions of the enlisted factors. Categories may belong to the data, event/process or environment aspect of the system. They are classified according to conceptual, logical and physical levels. In short: analysts are able to design their own designer system by SYDES.

On the other hand, some categories, properties and conventional values are predefined in SYDES. E.g. entity, attribute, relationship, process and event are prespecified SYDES-categories. This means that analysts do not have to prepare their own designer system, if design-factors provided by SYDES were sufficient.

Design-quality control is the most valuable function of SYDES. Many design criteria are built into the system. These are validated at entry of design information and by separately run analysis programs. We apply

mathematical and semantical evaluation methods. Normalform analysis is one of these. It is employed in a much revised form. The objective of this paper is to present our normalization process, which is closely coupled to the tasks of process-design.

We had a good reason to revise normalforms and normalization. Our first design-aid, SZIAM generated 3NF relations from attribute functional dependencies (FDs). We have used a normalform-synthesis method. (SZIAM has been licensed by IBM). The tool was applied for several large data-modelling tasks (including over 2000 attributes). We had some acceptable results. However, the method of synthesis proved to be very inefficient at such scales. We also had to conclude that data-design is an half-eyed giant, a Cyclops without a parallel process-modelling. The latter is required to capture more semantic meaning of data.

For the above reasons, the synthesis line was dropped. We have prepared a new product, ADAM & EVA. ADAM stands for analytic data modelling. This part of the tool supported normalform-decomposition (4NF). The other subsystem, EVA helped designers to define event-activity networks. The two functions were closely coupled. The product was double-sided in the sense that both a specially extended relational model and a similarly extended entity-relationship model could be prepared by its aid. (For some of virtues, the product has been licensed by BMW AG., FRG.)

We applied a particular decomposition method, which preserved some of the virtues of data-synthesis. Theory was again followed by practice. ADAM & EVA was used in several large applications. The conventional NF-decomposition have failed in many cases. It produced imcomplete, i.e. not connective data-designs. This fact was mostly due to incorrect semantic interpretation of data. We had to realize that data-modelling is rather a semantical than a mathematical endeavour. The next section shows, how our thoughts of normalforms/normalization had been changed.

2. The Nature of Normalforms and Normalization

Information system design-aids may help us to draw nice pictures, like entity-relationship diagrams. They can support management of design-information by 'meta-databases' or 'information resource dictionaries.' Nice tools guide the analysts through the complex design-process. All these tasks are inherent to a design-aid. Nevertheless, we intend to think that quality-assurance must be the most important function of CASE-tools.

Design-products must represent reality correctly. They have to be non-redundant, complete and unambigouos. CASE-tools should include quality-control processes as to ensure specification of optimum designs. We have found that normalforms were powerful means for design quality checking. An appropriate normalization process leads to a proper data-design. At the same time, it may improve the efficiency of process-design as well.

The nature of normalforms and that of normalization should be revised, before these concepts are used for quality-assurance of design-products. A very simple, but tricky example highlights the present problems of normalization.

In 1984, we had to face a design-tool applying the user-view integration approach. This data-design method assumes that the 'global' structure of a database can be deduced from 'private' data requirements of end-users. We have presented two simple views to that tool. (Please, observe our notation. Entity-types or relations, are shown in capital. Primary keys of data units are underlined. Parts of composite keys are connected by '+'. Attributes of entities are enlisted in paranthesis.)

(2.1) a/ ORDER (Order-no, Product-no, Quantity-ordered)
b/ ORDER (Order-no+Product-no, Quantity-ordered)

Five analysts have worked on this simple case, envolving not more than four presented concepts, for two hours - without success. The design-aid has failed. Why?

Two basic approaches are applied for normalization: decomposition and synthesis. Let us see, how far we get with any of those.

Both views are in correct normalforms (they are, as we shall see), so they cannot be normalized by decomposition. One may want to unite these two relations into one. The effort is useless: the keys are different. So, let us rename ORDER of b/ to ORDER-1! That would not help, either. The design of (2.1) suggests, that its relations are connectable by Order-no as a foreign-key (cp. 'referential integrity'). They are not.

Normalform synthesis leads to an even greater disgrace. It is based on mechanical normalform-rules. Quantity-ordered is defined by Order-no (View a/). It depends on Order-no+Product-no (View b/), as well. These two statements suggest the partial dependency of Quantity-ordered. Thus View a/ will be the only relation resulted from synthesis.

Let us have a different look at the case of (2.1). We ask the end-user about the meaning of those four concepts. In other words, a semantical

analysis is executed. After a few simple questions, we shall understand that there are two kinds of orders: customer- and purchase-orders. The concepts in View a/ and View b/ are homonyms. Customer-orders (View a/) are always related to a single product, while purchase-orders (View b/) may have several items.

Conclusion: in terms of normalforms both views are perfect. None of them has to be or can be normalized. However, homonyms must be eliminated. An unambigouos capturing of the two views would result in the next design:

(2.2) CUSTOMER-ORDER (C-Order-no, Product-no, Quantity-ordered)
PURCHASE-ORDER (P-Order-no+Product-no, Quantity-ordered)

Nicely-cut examples are presented in publications. Life is more complex. Design-tasks may envolve hundreds or even thousands of concepts. An army of designers and a great bunch of end-users are working at the design. Communications falter. End-users do not immediately capture the meaning of 'dependencies', and they may make erroneous statements. Analysts are working in separate groups, so synonyms and homonyms are hard to avoid in the overall design. Contradictions, misunderstandings and even lies are parts of the game...

Design-tools have to help analysts in discovering all discrepancies. It took only a few seconds with our ADAM & EVA tool to solve the problem of (2.2). Please, observe, that this was not a fancy-case. We have faced this very situation at the information modelling of a suit-factory.

Having applied data-modelling techniques at several dozens of companies and institutions, we came to the following conclusions:

. Normalization and normalforms are mathematical notions; quantitative measures to improve a data-design. They are of no use, if the basis of normalization was not free of homonym and synonym concepts. The use of normalform decomposition or synthesis must be preceeded by a very careful semantical analysis, i.e. clarification of concepts.

. Normalform analysis may show, that preliminary statements of concepts were of bad quality. Like in case of (2.1). We have executed a special normalization. We came to the conclusion that nothing is wrong with normalforms; the concepts themselves had been badly formulated. A problem of quantitative nature has called our attention to the real qualitative trouble.

. Data-design is an iterative process, which follows the 'se-ma-for' principle. Semantics first, mathematics next, and the former again.

Our previous findings may be of no great news to some experts. However, we have met many designers, who had applied normalization principles mechanically. We have read many 'classics' of normalization, too. They seem to neglect the aspect of semantics completely, as we shall prove in the next section.

3. Incompleteness of Normalforms

Normalforms are said to be complete in the sense, that 5NF covers the most important kinds of dependencies. 5NF is the ultimate normalform. That we do not doubt. However, we have discovered that the series of NFs had been incompletely defined internally (!). This statement is easily proved by the following decision-table. Before its interpretation, we hasten to declare that we do not intend to implement new NFs into the present cavalcade of dependencies. We just have some semantical remarks.

(3.1)

	1	2	3	4	5	6	7	8
A --> B	P	P	P	P	D	D	D	D
B --> C	P	P	D	D	P	P	D	D
A --> C	P	D	P	D	P	D	P	D

Three functional dependencies (-->) of three attributes (ABC) are shown. 'P' stands for a trivial dependency. In this case a primary-key defines its own part. Like A=(X+Y) --> B=(X). 'D' shows a normal dependency. The key defines a descriptive, non-key attribute.

It is easy to see, that Rule 8 stands for transitive dependency (3NF). Rule 4 explains partial definition (2NF), and Rule 7 covers key-breaking (BCNF). Rule 2 is impossible: if C is part of B, which is contained in A than C must be a key-part of A. But how should the other rules be interpreted? We have found no reasonable treatment of them in the available literature. (Note: This table was composed back in 198Ø.)

There are two cases. If it is not important, whether a defined attribute is part of the key (P) or not (D), than there is no sense to make any distinction among 2NF, 3NF and BCNF. In all these rules dependency A --> C is transitive (it is not partial or key-breaking). However, if this distinction makes sense, than the series of normalforms is, indeed, internally incomplete. It does not cover four rules of Table (3.1). We believe that these remaining rules must be treated separately, if a correct semantic interpretation (semafor-principle) is to be applied.

Examination of the remaining dependency-structures (rules) follow.

Rule 1 is a set of trivial dependencies. It would be a mistake to handle this case as transitivity, partial-dependency or key-breaking. (This particular rule implies all of these three normalform-problems. A --> C is transitive. A=(X+Y+Z) --> B=(X+Y) --> C=(X) is partial. The last part of this series is key-breaking.) An example:

(3.2) CITY (Country-code+District-code+City-code)
City-id=Country-code+District-code+City-code

Rule 6 may be considered as a 'group-dependency'. A key defines a group of attributes, which defines its parts. A --> B=(X+Y) --> C=(X). There is no way to resolve this set of dependencies. However, one may ask the question: is C=(X) semantically identical to the part of B=(X+Y)?

(3.3) ACCIDENTS (Accident-no, Date, Month)
Date=Year+Month+Day

Note: A relation is supposed to contain elementary attributes only. This 'law' is often neglected in data-designs. Thus the question: 'Is Month identical to the part of Date?' is a crucial one.

We have named Rule 5 as 'intersection', since two keys have a common part and one defines the other. A=(X+Y) --> B=(X+Z) --> C=(X). An incorrect normalization would ban A --> B. (Note, that A --> C is not transitive, it is trivial. B --> C cannot be eliminated, either.)

(3.4) DISPO-ITEM (Dispo-item-id, Order-item-id, Quantity)
Dispo-item-id=Dispo-id+Product-id
Order-item-id=Order-id+Product-id

The relational model does not recognize attribute-groups. This causes a lot of troubles. Order-item-id should be a foreing-key (cp. referential integrity), but in the 'orthodox' approach there is no way to define it. Thus the question arises: How to connect dispo-items to order-items? Rule 3 is called by us as 'hierarchical key', because one part of the key defines another part. A=(X+Y) --> B(X) --> C(Y). This is neither the 'normal' key-breaking dependency, nor a partial one. There is one only way to resolve this problem: to change the key. Normalization cannot help. A semantical solution is required.

Observe that this last case is the same as the one provided in (2.1). Our design-aid was able to call our attention to a semantical problem, because it had examined hierarchical keys and it have noticed a trouble of mathematical nature. Our tool handles the rules of (3.3) and (3.4) in a similar, semantically based fashion.

In summary: One may face situations, in which normalization-routines cannot help. However, they may call the attention to deeper troubles of semantical roots. Having eliminated homonyms, synonyms, incorrect keys, one may redo normalization according to the 'semafor' principle. This idea is explained in the next section.

4. Semantical Normalization

One may read several publications about 'linear-time' normalization processes, trying to overcome 'quadratic' or 'exponentional' routines. This is very nice: it is good to have optimal normalization algorithms. Unfortunately, they are not up to the issue.

Some analysts pretend to believe, that one has a nicely defined set of relations and attributes, so let us apply a good normalization algorithm and then we shall have the appropriate data-design. Some design-tools follow this principle. They remove incorrect dependencies automatically (decomposition-based tools) or they do not allow speecification of a bad dependency (aids of synthesis). Again, they are not up to the issue.

The two key-points of normalization are interpretation and design time-frame. In nice, small, academic examples one has a predefined set of concepts. In reality, data-design may take several months, and one part of the system must work, before another part is designed. There is nothing like a 'universal relation' or 'minimum cover'. The analyst must apply a 'co-normalization', trying to add new fractions to an already existing database in the best possible fashion. This was stated for the time dimension. Now let us examine the interpretation aspect.

If an A --> B --> C dependency occurs and the A --> C dependency is entered/discovered, than most of the normalization tools would remove the latter definition automatically. Our first design-tools have worked according to this logic, too. Then we have found, that well over 60% of normalform problems resolved by the design-aid were due to semantical misinterpretations. In other words: in terms of mathematics the third (A --> C) dependency was incorrect, though actually - in semantical terms - either A --> B, or B --> C had been falsely specified. (This is plain algebra. There are three dependencies. The chances that the third one is incorrect, are at 33%.)

Data-designs grow because of the time-frame. New attributes and new relations are added to a working environment. We do not endeavour here

to explain, how and why does the sequence of such additions influence the mathematical result of normalization. It is enough to state that the present mathematical basis of data-design is incapable to cope with this problem of growth.

In summary: Conventional normalization algorithms are not acceptable. They are not based on a semantically sound set of concepts, or at least they do not seem to care about them. Normalization is a bunch of very mechanical routines, with no back-loops for semantical corrections.

In SYDES, a different method is practiced. Designers may define their entities (relations), data-items (domains) and their connections (attributes of relations) 'by heart'. Having defined the basic items, the analyst may run analysis-routines of SYDES. These will inform her or him about possible normalform problems. Not just about the ones, known from the literature. SYDES works on the basis of the (3.1) table. The designer may follow three routes:

. Consider NF problems as quantitative troubles. When A --> B --> C, attribute C is to be removed from the relation identified by A. This is the 'orthodox' approach.

. Ignore NF problems. One should recognize that while a 2NF data-unit is worse than a 3NF relation in terms of joins, it may have many virtues in other design aspects. This is the practical approach.

. Consider NF problems as qualitative troubles. When A --> B --> C, any of the functional definitions (including A --> C) may be resulted from incorrect interpretations. This is the semantical approach.

This is the essence of SYDES-logic. Tools cannot examine the qualities of concepts defined by human beings. Nevertheless, when constructed for this purpose, they can call our attention to semantical problems by evaluating quantities. This is the essence of semantical normalization. The basis of such a normalization process is described below.

5. The Concept of 'Threads'

Conventional NF-decomposition methods cannot be used for database design. The unit of normalization is a suggested relation, without any structural reference to similar such units. The problem in (2.1) cannot be solved by this procedure. The notion of referential integrity will not help either. One question is, whether individual relations are

well-defined or not. A second matter of investigation is, if the whole set of defined relations were optimally designed. A very simple example highlights this problem:

(5.1) CUSTOMER (Customer-id, Customer-address)
ORDER (Order-no, Customer-name)

Both relations of (5.1) are in 'perfect' normalform. However, the design as a whole is a complete mess.

Having recognized the shortcomings of decomposition, new ideas arised, like covers and universal-relations. They are not really useful. Let us suppose, that one has 1000 attributes (only). Stating dependencies among them would require about 500 thousand investigations, if one wanted to find all possible dependencies. This work cannot be done.

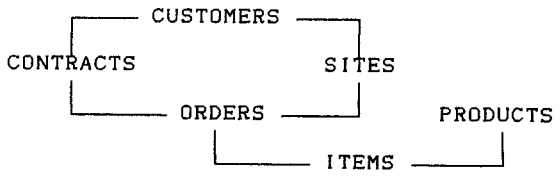
The unit of decomposition (single relations) and that of synthesis (universal relation, cover) are not acceptable. We seem to be stuck.

A foreign-key in a relation is an attribute of that relation, which refers to the key of another relation. The key of the first relation functionally defines all of the attributes of the second one. Like Order-no defines all customer attributes through Customer-id in (5.2).

(5.2) CUSTOMER (Customer-id, Customer-name, Customer-address)
ORDER (Order-no, Customer-id)

Please, observe, that a foreign-key gives rise to a hierarchy. The real-world phenomena to be represented by information are referred to as entities. They are classified to entity types. ORDER and CUSTOMER in (5.2) are entity-types represented by relations. By nature, they are in hierarchical connection: a customer may have many orders, but each order belongs to a single customer. This fact is reflected in the design by Customer-id of ORDER pointing to CUSTOMER.

(5.3)



Two entity-types may be connected directly. They may have an indirect hierarchical relationship, or they may be independent of one another. In (5.3), ORDERS is directly related to CONTRACTS and indirectly to CUSTOMERS, while it is independent from PRODUCTS. This means that ORDERS

has a direct reference to CONTRACTS (Contract-no); an indirect coupling to CUSTOMERS (CONTRACTS contains Customer-no) and no connection is made to PRODUCTS (none of the superordinates of ORDERS has Product-no).

In SYDES, we use the 'thread' term. A thread is a bottom-up directed line of entities leading from the lowest entity to the highest one. In case of (5.3), we have three threads. Items/orders/contracts/customers; items/orders/sites/customers and items/products. Threads are represented by keys of entities. They are nothing but chains of dependencies.

A thread is an intermediate unit. It is neither a single relation, as used in normalform-decomposition, nor a universal one, applied in the logic of normalform-synthesis. As we had seen, none of these methods can be used in a real practical data-design. The question is, what is the trade-off of implementing the thread-concept.

All kinds of incorrect (e.g. transitive, partial etc.) dependencies are easy to discover along the threads. Before such a normalization, cycles must be eliminated. The following set of dependencies is circular, so it is a cycle: A --> B --> C --> A. (Note, that some designers would see a cycle in (5.3), too. That example does not have a cycle.) The nature of entity-relationships are also entered to threads. Mandatory and optional connections (strong and weak FDs), partial and total as well as subtype relationships may be specified. This helps us to examine connectivity of the data-model. (Whether all entities are accessible from the others.)

Normalization of keys is the first task to be executed, because of possible cycles. Any of the dependencies may be incorrect in a cycle. Not necessarily the one, at which the cycle had been closed. Human reinterpretation is required. Dependencies must not be removed by a tool automatically. Having eliminated occasional cycles, transitive, partial and other problems are searched for. SYDES manages composite keys both as singular units ('A') and as collections ('A=X+..+Z'). A composite key may define another one, unlike in other normalization algorithms. Thus key-breaking and group dependencies, intersections and hierarchical-keys can be discovered.

Normalization of keys is followed by normalization of other attributes. Threads cannot be used directly to examine occasional FDs between items on parallel branches, such as ORDERS and PRODUCTS in (5.3). This may seem to be a major shortcoming. It is in theory, but not in practice. We have experimented with threads for quite a long time. We have found that at most 13% of 'hidden' dependencies had been undetectable by our process at very large data-models. These dependencies, and many more,

could not have been detected by conventional normalization either.

One may argue, that such a normalization process is not linear. It is not. However, normalization is not a one-time effort. The time required for normalization is T * E. 'T' stands for the time of one execution of the analysis and 'E' shows the number of iterations. Normalform problems are mostly due to human failures, like semantic misinterpretations. If an incorrect dependency occurs, a semantical analysis is required. The really bad dependency must be removed, not the one suggested by a tool. Removal of a dependency or migration of an attribute from one relation to another has the consequence that normalization must be repeated.

Threads are very nice means to find the proper place of attributes in one only analysis run. Thus 'T' may be higher as in linear-processes, but 'E' is definitely lower. This is explained by an example:

(5.4) A --> B --> C --> D --> e and A --> e

Key-attributes are shown in upper-case. We have a single descriptor attribute 'e'. When having 3000 attributes, a universal relation is out of question. When applying decomposition, 'e' would migrate from A to B, then from B to C, then from C to D. This is the point, at which its transitivity can be detected. Four iterations ('E'=4) were required. This transitivity is easy to discover at once when using threads.

Threads are redundant. They are overlapping. This is a storage problem only. It has nothing to do with time required for normalization. Common subthreads are analyzed by SYDES only once in one walk-through.

Conclusion: Threads may not be the most efficient means for a single execution of a normalization algorithm. Howewer, they are very useful for semantic normalization to reduce number of runs. In addition, our heuristic process increases the overall efficiency of the desig-effort and the overall optimality of the design-product. This idea is developed in the last section.

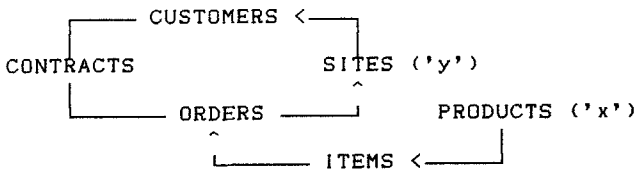
6. Threads in Process-Modelling

We cannot agree to the 'data-design first' principle. In nice, small applications that route may be followed. It is unusable in large projects. Data- and process-modelling are parallel efforts, which are coupled at a particular phase of the development.

One of the design subtasks of this coupling is definition of so called navigation pathes. A navigation path is a set of entities managed by a particular process-unit. The set is ordered in the sequence of accesses. At a more detailed level of design, specification of such navigations include access-mode, type of operation executed on the entities, set of attributes taken from given entities and their usage-mode. Navigation pathes can be supplemented with access- and hit-ratio information. These information can give rise to automatic prototyping of programs. At the same time, definition of pathes is a good control for data-modelling.

Let us see a simple example related to (5.3). The task of the process is as follows:
"Find all those customers, who have ordered product 'x' for their sites located at 'y'." A possible navigation path for this query is in (6.1):

(6.1)



Such pathes are inherent parts of a process-design, so they have to be stored in the design-dictionary. As to define them, one must have the network of entities at hand. This is required for analysis reasons. One must investigate the next questions:

- Are entities managed by the process specified at all?
- Are they internally connectable, or external control is required?
- What is the reasonable sequence of entity management?
- Are there any possible alternate routes?
- What is the direction of access?

Navigation may be directed upwards and downwards. The entity, at which this direction changes, is referred to as an inflexion point. In case of (6.1) ITEMS is such.

How to manage the required entity-relationship 'diagram'? Storing all possible pathes would be nonsense in case of several dozens or hundreds of entities. However, such pathes are very easy to project from threads.

One can specify starting (PRODUCTS), inflection (ITEMS), intermediary (ORDERS, SITES) and closing (CUSTOMERS) points of navigation pathes.

We have a Hungarian saying: "Two flies for one flap". Threads are very useful in process-design. Covers and universal relations are not. Thus data-normalization and 'process-normalization' can be coupled in SYDES.

Processes themselves can and must be 'normalized'. Process structures may also be ambigouos, incomplete or redundant. Some of the entity operations (addition, deletion, modification) may be missing. Some could have been defined duplicately. Parallelled by statements of operation-type, thread-manipulation may give us a convenient basis for analysis of entity life-cycles. When two processes are mapped to threads in the same way, i.e. they manage the same entity-types in the same sequence, the analyst may consider to design of one process instead of two.

Constraints are very important notions in modelling information systems. Normalforms themselves are particular kinds of them, but other types of constraints should also be implemented. Conditional process-branches, entity-association (not just referential) rules, subtyping of entities, appropriate usage of role-names are just a few of them to mention. These constraints pertain to a pair, a chain or a special subset of entities, and not just to one of them. These constraints are easy to define and validate by the aid of threads.

SYDES allows for definition of external input and output of a process as virtual relations. These are implicit starting and ending leaves of a navigation path. Data-flows can be represented in this way. Coupled to entities of threads, a most important constraint can be validated. A process must be able to provide its output from its input and the items of the navigation path. SYDES supports an HIPO-like process-analysis.

7. Conclusions

CASE-tools should envolve relational normalforms for analysis of data-structures. Conventional normalization algorithms are better to avoid. Completeness of NF-concepts must be revised and groups have to be used at least in the analysis and design phases. Normalization is to follow the 'semafor' principle. Threads are very powerful constructs to reduce the number of iterations during the design process. They lend themselves for an easy navigation definition as well as for stating constraints. They proved to raise the overall efficiency of the development effort.