Why Normalization Failed to Become the Ultimate Guide for Database Designers?

Marin Fotache

Alexandru Ioan Cuza University of Iasi, Romania

Faculty of Economics and Business Administration

Dept. of Business Information Systems

Blvd. Carol I, nr. 22, Iasi, 700505, Romania

Tel: + 40 744 497 654, Fax: + 40 232 217 000

fotache@uaic.ro

ABSTRACT

With an impressive theoretical foundation, normalization was supposed to bring rigor and relevance into such a slippery domain as database design is. Almost every database textbook treats normalization in a certain extent, usually suggesting that the topic is so clear and consolidated that it does not deserve deeper discussions. But the reality is completely different. After more than three decades, normalization not only has lost much of its interest in the research papers, but also is still looking for practitioners to apply it effectively. Despite the vast amount of database literature, comprehensive books illustrating the application of normalization to effective real-world applications are still waited. This paper reflects the point of

view of an Information Systems academic who incidentally has been for almost twenty years a practitioner in developing database applications. It outlines the main weaknesses of normalization and offers some explanations about the failure of a generous framework in becoming the so much needed universal guide for database designers. Practitioners might be interested in finding out (or confirming) some of the normalization misformulations, misinterpretations, inconsistencies and fallacies. Theorists could find useful the presentation of some issues where the normalization theory was proved to be inadequate, not relevant, or source of confusion.

Keywords: database design, normalization, functional dependencies, multi-valued dependencies, normal forms

INTRODUCTION

Something is wrong with normalization. Not only because it is old and ugly. Apparently, almost every database author, guru or not, and teacher seems to have it at his/her fingertips, but when deepening the discussion, it becomes obvious that there is a swampy area, where mathematics is the best shelter for running away from the real world database design problems. It also seems that normalization topics have become too obsolete for the scientific community. Besides a number of articles trying to establish the normalization principles for XML databases (!), apparently there is nothing new and valuable to be said in the field. For a researcher it is quite embarrassing to confess that he/she is interested in normalization. If not adding quickly

into discussion some subjects, as UML, XML, ontology, semantic web, agile development, etc., that researcher risks to be out of the researchers camp.

Database normalization for practitioners seems to turn into an oxymoron. Normalization is now famous for not establishing many links with real world applications database design. Who is the culprit for this state-of-the-art and state-of-the-practice ? With no intention to blame someone in particular (but many, in general!), one can enumerate database authors, relational gurus, academics (like me), and last, but not least, practitioners (also like me) who haven't annoyed the theorist about their expectations concerning normalization and its relevance for the real applications. Beyond research papers saving database course tenures, it has been a great "achievement" to turn database normalization into nothing but a boring theory, a(n extra) way of torturing Computer Science/Information Systems students. Muller (1999, 321) pointed it out: "It is possible to write an entire chapter or book on normalization, though hard to make it fun to read".

As the whole relational model, normalization theory was initially firmly designed - a sound theoretical framework based on functional, multivalued and join dependencies among attributes, theorems of decomposition, inference rules, closure, minimal cover and normal forms. Unfortunately, the theoretical achievements have not been followed up by practical concerns and debates about the extent to which practitioners can apply normalization to design appropriate database schemas for large applications, and for dealing with complex relationships among attributes.

Moreover, some popular books dedicated to the topic of database design do not use normalization at all (e.g. Hernandez, 1997) or, if they do, it is only to a very little extent (e.g. Muller, 1999; Stephens and Plew, 2001; Carlis & Maguire, 2001; Halpin, 2001).

The paper will try to identify the main normalization approaches in some of the best known books and articles dedicated to database design, presenting some of the normalization difficulties and traps, and the way many texts failed in explaining what is useful and what is not in applying normalization in real-world applications. It is not a normalization tutorial, and the intended readers must have a minimal background in functional/multivalued dependencies and the normal forms, or at least could follow the suggested references for details.

A SURVEY OF NORMALIZATION

A relational database can take the form of a single huge relation with tens, hundreds or thousands of attributes and hardly countable tuples. But having a single relation within a database schema usually means an unacceptable redundancy, and a considerable source of insert, update, and delete anomalies (Date, 2004; Elmasri & Navathe, 2000; Garcia-Molina, Ullman, & Widom, 2002).

In order to fix some redundancies and anomalies, E.F. Codd (1972) proposed three normal forms, 1NF, 2NF and 3NF, achieved through loss-less decompositions (with dependency preservation) of initial universal relation into smaller relations, based on functional dependencies (FDs). Shortly afterwards, Heath and then Boyce and Codd (1974) identified some inadequacies of the 3NF definition, therefore they redefined 3NF and renamed it Boyce-Codd Normal Form – BCNF. Fagin (1977) was first to describe the fourth normal form, 4NF, based on multivalued dependencies – MVDs (Zaniolo; Fagin; Delobel), as well as the fifth normal form, 5NF (Fagin, 1979) - projection-join normal form (PJ/NF) -, which is based on join dependencies (JDs).

Well-known database books (e.g. Date, 1986; Elmasri & Navathe, 2000; Date, 2004) and articles (Zaniolo & Melkanoff, 1981; Kent, 1983; Smith, 1985) present, in various degrees, each step of normalization, from 1NF until 5NF. To be in 1NF, a relation must contain atomic values only (Date, 1986, p.362). A relation in 2NF does not contain any partial FD, whereas in 3FN every relation has no transitive FDs. In the case of BCNF the basci constraint for a relation is: every determinant must be a candidate key. To be in 4NF, all relation MVDs must be, in fact, FDs. A relation R, in order to be in 5NF, requires that every join dependency in R to be a consequence of the candidate keys of R.

The least attractive steps concern BCNF, 4FN and 5FN. Usually, the books stop with 3FN, or, at best, with BCNF. In some database books, MVD/4NF examples are not properly explained (Muller, 1999; O'Neil & O'Neil, 2001, Stephens and Plew, 2001; Simsion, 2001; Connolly & Begg, 2002), or not exemplified enough (O'Neil & O'Neil, 2001; Connolly & Begg, 2002; Muller, 1999) or not even remembered (Connoly & Begg, 2000). Similarly, join dependencies (JD)/5NF) frequently are not clearly explained (Muller, 1999; Pratt & Adamski, 1991) not exemplified enough (Connolly & Begg, 2002; Elmasri & Navathe, 2000), just remembered (Stephens and Plew, 2001), or even not discussed at all (Connoly & Begg, 2000; O'Neil & O'Neil, 2001; Garcia-Molina, Ullman, & Widom, 2002; Harrington, 2002).

Margaret Wu noticed that many academics and authors neglect 4FN and 5FN because of practical irrelevance. Examining company databases, Wu's survey revealed violations of 4NF in more than 20% of the investigated cases, but none of the schema violated 5NF requirements (Wu, 1992).

Besides the five normal forms, authors have challenged database community with interesting propositions, some concerning improved definition of classical normal forms (e.g. Ling, Tompa, & Kameda, 1981; Date & Fagin, 1992), others introducing new types of normal forms, such as

Zaniolio's *Elementary Key Normal Form* (Zaniolo, 1982) and Vincent's *Key-Complete Normal Form* (Vincent, 1998). Perhaps the most notable is *the Domain-Key Normal Form*, whose author is Fagin (1981). DKNF is built on three issues: key, constraint and domain. A relation fulfills DKNF requirements if, and only if, every constraint on the relation is a logical consequence of the domain constraints and the key constraints that apply to that relation.

Based on FDs and inclusion dependencies (Fagin, 1981; Casanova & Fagin, 1982; Beeri & Korth, 1982; Mitchell, 1983), Ling & Goh (1992) proposed the *Inclusion Normal Form* accomplished by extending the *Deletion Normalization Algorithm* described in Ling, Tompa & Kameda (1981). Mannila and Raiha (1986), as well as Levene and Vincent (2000), advocate for *Inclusion Dependency Normal Form*, a normal form combining BCNF with the restriction on the inclusion dependencies that must be non-circular and key-based.

Also Date (2004, 761-763), in the 8th edition of his prominent book, proposes the *sixth normal form* which he defines for solving temporal issues in database design. Although seductive to many theorists, these normal forms have not been accepted by practitioners mainly because of the lack of case studies proving their strengths and utility.

In order to accomplish the task of bringing the database into higher normal forms, two basic approaches were proposed. The decomposition approach - originated in (Codd, 1972) - outsets with an initial big (sometimes huge) relation containing all the database attributes and, through the elimination of repeating groups, partial, transitive, multivalued and join dependencies, splits up recursively the initial relation – called usually the Universal Relation – into smaller relations. With the synthesis approach (Bernstein, 1976) relations are built based on the set of dependencies among the database attributes.

QUESTIONING THE PLACE OF NORMALIZATION IN DATABASE DESIGN METHODOLOGIES

In the early days of normalization, the relational model was just theoretical, so database designers could not implement the database schema into any DBMS. During the 1980's, when relational database management systems became successful on the market, Chen's *Entity-Relationship* model (Chen, 1976), later enhanced to *Extended Entity-Relationship* model (Teorey, Yang, & Fry, 1986), was gradually adopted by practitioners because it was considered more natural to database modeling, and closer to real world application needs. In fact, most of the texts and papers still recommended starting logical design with E-R model and subsequently converting E-R schema into relational tables, following some basic rules (Chen, 1976; Teorey, Yang, & Fry, 1986; Markowitz & Shoshani, 1992; Teorey, 1999, pp.79-93; Connoly & Begg, 2000, pp.159-180 ; O'Neil & O'Neil, 2001, pp.334-338; Anthony & Batra, 2002; Garcia-Molina, Ullman, & Widom, 2002, pp.65-75).

The fact that E-R model has a rather different view of reality than the relational model introduces a discontinuity in the process of database design. O'Neil and O'Neil (2001, 352) accepted that "normalization is another approach to logical design of a relational database, which seems to share little with E-R model". Usually, CASE products automatically generate relational tables from E-R diagrams (Hainaut, Cadelli, Decuyper, & Marchand, 1992; Rosenthal & Reiner, 1994), but mapping an E-R schema to a relational schema, although attainable without exhausting efforts, is quite unnatural. After all, what do "validate user views trough normalization" (Fleming & vonHalle, 1989) or "validating relations using normalization" mean ? (Connoly & Begg, 2002, pp.455-456). Sometimes, after normalization, the E-R schema must be reverse modified, but it is almost impossible to find authors explaining how to do that, even the

converse way of generating E-R diagrams based on functional dependencies was described by Conception & Vellafuerte (1990).

Connoly and Begg (2000, 102) synthesize the two general perspectives: "Normalization can be used in database design in two ways: the first is to use normalization as a bottom-up approach to database design; the second is to use normalization in conjunction to with ER modeling". This basic statement must be corrected with the assertion that normalization can help in building a database schema from scratch using not only a bottom-up (synthesis) approach, but also a top-down (decomposition) one.

Teorey (1999, 5-6) identifies four steps of logical design: ER modeling, view integration, transforming of the ER model to SQL tables, and finally normalization of tables. Pratt and Adamski (1991), Teorey (1999) apply normalization starting from a relational structure, no matter how the structure was obtained. For them, normalization is not concerned so much with the initial schema design, but "with analysis the design of relational database to see whether is bad". Even more, Stephens and Plew (2001, 186) shrink normalization to "the process of reducing the redundancy of data in relational database".

If normalization is applied on tables yielded through ER conversion, then why debating the Universal Relation, which, by definition, contains all the database attributes ? Moreover, is normalization suitable for modeling ? Could it serve as first step of logical design ? Halpin (2001, 628) argues that the use of normalization alone is inadequate as a design method, but, nevertheless, using it as a check on conceptual modeling may be at times be helpful.

Nowadays it is accepted that ORM (Object-Role Modeling), Object Oriented (OO) and other post-OO methodologies, such as Agile Modeling, Agile Development (Erickson, Lyytinen, & Siau, 2005) are most appropriate not only for programming, but also for analyzing and designing information systems, including, of course, database design. The problem for database

designers is that OO models are even more different from relational "philosophy" than E-R model. Although some authors have prescribed the basic rules for mapping ORM and OO schemas to relational schemas (Halpin, 2001, pp. 403-454; Garcia-Molina, Ullman, & Widom, 2002, pp.155-164; Muller, 1999, pp.271-344), translating the objects into relational tables is not a trivial problem.

UML diagramming methodology becomes increasingly accepted (Muller, 1999; Halpin, 2001), its strengths and limits being acknowledged (Siau, Erickson, & Lee, 2005). Currently it is almost impossible to imagine the analysis/design of an information system without UML support, though the appropriateness of UML for analysis is sometimes questionable (Hay, 1999; Halpin, 2001, p.250). Furthermore, some authors warn about UML fever "pandemy" (Bell, 2004).

If analysis and design end up with classes, objects, methods etc., why not implement them directly as they are, without conversion into tables ? Because at his tome "pure" OO databases cover less than 6% of the DB market, and despite the suitability of OO analysis and design "products" (diagrams), their required conversion into relational tables could be tedious.

But one day Halpin's assertion could become generaly true, no matter the type of conceptual schema modeling approach: "In the past, in place of conceptual modeling, many relational designers used a technique known as normalization. Some still use this, even if only to refine the logical design mapped from conceptual one. If an ORM conceptual schema is correct, then the table design resulting from Rmap is already fully normalized, so ORM modelers don't actually need to learn about normalization to get their designs correct" (Halpin, 2001, 627). Well, let's say good-bye to normalization and keep a moment of silence for its remembrance !

TOO MUCH FORMALISM (MATHEMATICS) ON PRACTITIONERS' SHOULDERS

As Kent (1983b) pointed out, the hallmark of relational theory is rigor, the objects of that theory being defined with mathematical precision. Sometimes too much rigor in theory can lead to annoying problems in facing reality. And, certainly, database normalization deals with reality, being a methodology employed for designing acceptable schema, in order to catch and manage the relevant information for the process, transactions, and operations of the application developed. Even in the case of design methodologies considered more flexible, theoretical complexity might not accurately predict practical complexity (Siau, Erickson, & Lee, 2005).

Simsion (2001, 35) warns that there are plenty of texts and papers on normalization, but the reader must usually take a fairy formal, mathematical approach. On the same line, Biskup (1998) argues that theoretical investigations have accumulated many formal notions and theorems on database schema design but, unfortunately, theory apparently does not have much impact on practice yet. I totally share Simsion's (2001,38) sincere assertion: "it is very easy to become lost in mathematical terminology and proofs, and miss the essential simplicity of the technique".

Taking, for example, the relation minimal cover, which is a set of FDs from which all other FDs can be derived through formal properties (such as Armstrong's axioms). Computing minimal cover has been considered important in database normalization theory, and consequently a number of algorithms have been proposed (Zaniolo & Melkanoff, 1981; Garcia-Molina, Ullman, & Widom, 2002; O'Neil & O'Neil, 2001; pp.369-371). Similarly, closure of a set of FDs, which includes the FDs inferred or deduced from dependencies describing semantic relationships among the attributes of the relation has been subject of many algorithms and

techniques for (e.g. Diederich & Milton, 1988). Trying to apply them for small and medium application - not to speak about large applications - is a totally different story. The annoyance comes from the impossibility of founding papers or books in which these algorithms are actually applied for a practical purpose. So it is natural to question if database designers can use them.

Similarly, "sound-promising" (for designers) papers, such as *Synthesizing Independent Database Schemas* (Biskup, Dayal, & Bernstein, 1979), ended drowned in formulas.

Apart from few notable exceptions - (Kent, 1983a), (Smith, 1985), (Pratt, 1985), (Salzberg, 1986), (Simsion, 2001) - the vast majority of the texts and papers dedicated to normalization are formalistic in their nature. There is nothing wrong with mathematics. There is a general tendency – sometimes a desperate one - to embed mathematics in every paper for proving or at least suggesting that the research is done "scientifically". The real problem comes from the fact that almost nobody has tried to translate the theorems, lemmas, proofs, etc. into database designers language. So it's not surprising at all that most of the popular books on database design lack mathematics and theorems (Fleming & vonHalle, 1989; Hernandez, 1997; Teorey, 1999; Muller, 1999; Harrington, 2002), and designers have become interested in other formalisms like E-R, Object-Role Modeling, and UML.

DIFFICULTIES, INCONSISTENCIES, AND IRRELEVANCE IN NORMALIZATION LITERATURE

Some of the normalization concepts have been confusing from their early days, some have been restated many times, ending up in misconceptions and inappropriate use. How must be understood Universal Relation and what is its utility ? What is atomicity and do designers have

to avoid it ? How to deal with symmetric FDs and noncanonic transitive FDs ? Not all FDs that seem transitive are indeed so. Why do only few authors urge using inclusion dependencies, despite their essentiality in real world applications ? These are just a few issues that deserve a deeper discussion.

In other cases, ideas and models developed by theorists simply were not relevant to the practitioners: remember, for example, the Edmund Lien's nonflat, or hierarchical, view of relational databases (Lien, 1981). Combining the strengths of relational and hierarchical data models, the model also inherited the main problem of every tree model: when modeling a real-world application, organizing nodes of the hierarchy can be done in very different ways, depending on the interest and experience of the modeler. It is not always obvious how to start a hierarchy and notably and how to deepen the hierarchy's levels. Further data analysis may require completely different re-organizations of the tree. Other theories, such as *Nested Normal Form* (Ozsoyoglu & Yuan, 1987), really intimidated the non mathematics-addicted.

Recently confusion concerns the recycled notion of *partial functional dependency* which, despite its longstanding career, has been recently redefined as a FD that *almost holds* (see Matos & Grasser, 2004). The notoriety of "classical" partial FD would make almost unacceptable the new definition, and, perhaps, it would be a better idea to change its name in *probabilistic functional dependency* or *approximate FD*, as the paper title and section 2 suggest. Ilyas et al. use a similar concept but with different name – *soft functional dependency* – developing a tool (CORDS) for automatically discovering statistical correlations and soft functional dependencies between pairs of columns (Ilyas, Markl, Haas, Brown, & Aboulnaga, 2004). Mining into huge databases for identifying statistical links among attributes could be very fruitful in terms of relevant information extracted from the database. But let's make it clear. This type of mining operation

has almost nothing to do with database design, being allotted considerable time after database normalization (if the database was really normalized) and implementation.

How Useful Is the Universal Relation for Normalization ?

Practitioners generally consider Universal Relation (UR) the initial set of all the database attributes identified by the designer ("normalizer"). As stated, the concept of UR assumed that for a given set S of relations under consideration, there is in principle a single universal relation U such that each relation in the set S is a projection of U (Kent, 1981). The only clear initial assertion about UR concerned the fact it is a single relation; subsequently, some voices argued in favor of UR uniqueness for a database, or raised the question whether UR is subject to constraints or not, or if one can define UR by predicates (Kent, 1982b; Fagin, Mendelzon & Ullman, 1982).

Kent (1981; 1983b) underlined the difficulties induced by UR concept for many common practices in relational databases, stating that the UR model is an unsatisfactory model for relational theory and practice. On the contrary, Ullman (1982) defends the validity and relevance of UR concept, criticizing the arguments against UR. Actually, Ullman does not defend the idea of uniqueness of a UR for a given database - which he calls *Pure UR assumption* - , but advocates for the concept of *weak instances* in which the representative instance is defined as containing the maximum information that can be deduced at any given moment from a database, given a set of integrity constraints on the database (Maier, Ullman, & Vardi, 1984; Levene & Loizou, 1991).

Referring to UR, Atzeni and Parker (1982) reproached that "making assumptions (…) more for establishing certain results (such as the usefulness of 'decomposition' as a design

methodology) than for their appropriateness in modeling data, one must realize that the results may not be useful to database designers".

Other authors are more enthusiastic about the role played by UR in relational theory, arguing that whereas the relational model supports only *physical* data independence, UR offers more – *logical* data independence – by providing the database with an interface, which allows the user to view the database as if it were composed of a single relation (Levene & Loizou, 1991). But, let's be realistic, who could be interested in seeing entire database as a single relation ? Maybe in simple applications, single-view database might be relevant, but in the case of an ERP database, none of the users would like to see a table with thousands of columns (not to speak about the rows).

To database designers applying whatever of the decomposition or synthesis method, uniqueness of UR is not really an important issue. UR, composed of all the database attributes, represents just an outset for normalization. Ullmann (1982) remarked that UR does not actually exist, except in the user's mind. However, building UR is not as trivial as it seems to be, because some attributes must be merged, others renamed, and some eliminated or modified (Kent, 1981).

1NF and Atomicity. What Atomicity ?

First Normal Form (1NF) – along with atomicity as a fundamental conjecture for a relation to be in 1NF - has dramatically evolved over the years, not only as definition, but mostly as interpretation. While in first edition of his best-acknowledged book, Date (1975) stated: "Every value within a relation – i.e., each domain value in each tuple – is an atomic (non-decomposable) data item, e.g., a number or a character string" (Camps, 1996), in the most recent

edition "a relvar is in 1NF if and only if, in every legal value of that relvar, every tuple contains exactly one value for each attribute" (Date, 2004, p. 358). Former definition was interpreted as restricting each attribute to have only simple, nondecomposable values, whereas the latter is more flexible, each attribute value could be defined on a composite type.

Despite the fact that Codd (1970), accepted the idea of domains having relations as elements, and that relations could be defined on domains having relations as elements etc. in a recursive way, the vast majority of the texts and papers dedicated to relational database design has never taken seriously this idea. Actually, Codd himself induced this direction because in the same paper he acknowledged the desirability of eliminating nonsimple domains, and moreover, he called the procedure of eliminating nonsimple domains *normalization* ! As consequence, the general accepted idea was that in normalized relations, the elements of a domain are not permitted to be relations or sets themselves (Schmid & Swenson, 1975; Bernstein, 1976) even if other authors showed that it is quite possible for relational systems to deal with abstract data types for domains (Osborn & Heaven, 1986).

Strangely, Codd stated a more rigurous interpretation of atomicity only in 1979: "A domain is simple if all of its values are atomic (nondecomposable by the database management systems)" (Codd, 1979). Date argued that string, integer, and date values are decomposable by the DBMSs, still they are atomic ! (Date, 2005, 29-30).

Favorite subject of criticism by OO and Object-Relational (OR) data models supporters, Abstract/User Data Types have been finally included in the relational model (Darwen & Date, 1995). Now, types and domains are considered synonymous and interchangeable, and current relational literature tends to privilege, in a certain extent, the term *type* instead of *domain*. So the problem of accepting, within databases, no matter complex datatypes (including collections - sets, bags) is solved for the relational model. Date states quite clearly: "It is a very common

misconception that the relational model deals only with rather simple types: numbers, strings, perhaps dates and times, and not much else. Rather, relations can have attributes of *any type whatsoever* […] those types must be, and in fact they can be, as complex as we like […] types are orthogonal to tables" (Date, 2005, 36-37).

In the 1970's and 1980's a number of proposed models allowed repeating groups and relations to be tuple components. Perhaps the most famous was NF^2 (Non First Normal Form) or Nested-Relational (Makinouchi, 1977) which at that time was considered an extension of relational data model. To deal with repeating groups, relational algebra was enriched with two operators – *nest* and *unnest* – which transform 1NF relations into NF^2 relations and vice versa (Jaeschke & Schek, 1982). Initially Date & Co. reject the model, because it involved a major extension to the classical Relational Algebra (!) (Darwen & Date, 1995). Again, the idea of having *Relation-Valued Attributes* (RVA) has been finally accepted by the relational "purists" (Date, 2005, 31). RVA allow a flexible way of structuring data without giving up the power of relational model: the constraints mechanism and relational DDL and DML languages.

As further reading concerning atomicity, repeating groups, and 1NF, I suggest Date's texts (Date, 2003; Date, 2004; Date, 2005), and Pascal (2003) - for a severe criticism of "fallacies prevalent in the industry" -, but also Camps (1996) for an equally severe criticism of relational orthodoxists and their inconsistencies along the years.

To a large extent, database design textbooks and guides still recommend the atomicity of the attributes as basis for normalization (Teorey, 1999; Simsion, 2001). I share the opinion of considering atomicity of an attribute differently, depending on application needs. It seems astonishing, but it is perfectly acceptable that every attribute must be treated atomic or not contextually, according to the application requirements. In Date's words, "the notion of atomicity has no absolute meaning" (Date, 2005, 31).

A classic example is ADDRESS. Many company databases store address as an (atomic) string containing street name, number, building name, entrance, floor, and room. While address is necessary just for mailing interest or better person/customer identification, the ADRESS atomicity is reasonable. But for a company who supplies electrical energy it is necessary to split the address into distinct attributes: STREET, NO, BUILDING, FLOOR, ROOM. In the case of a major dysfunction at an energy distribution station, the company must know which streets, buildings, etc. will be affected by the failure.

A more relevant example concerns a library. The attributes are: ISBN, TITLE, PUBLISHER, PUBLICATIONYEAR, BookID, AUTHORS, KEYWORDS. Keywords are essentially for searching, so the database can be queried for extracting information such as the books containing chapters dedicated to SQL, relational algebra, normalization, etc. For a better search, from relational perspective, it is a good idea to "atomize" the KEYWORDS attribute, forcing a single value in a tuple, even that will bring a lot of redundancy in the database, redundancy expected to be eliminated in further normal forms. Still, "atomization" is not compulsory, because there are many functions for dealing with strings. Therefore, even there are many keywords for a single book, it is relatively simple to extract substrings matching a search pattern.

As for the BookID, there is a different story. Readers borrow an "occurrence" of a title. The librarian must know which readers have borrowed any specific book occurence. Because BookID is supposed to be a foreign key in a table which keeps tracks of all the book borrowings and returns, "atomization" of BookID is advisable. Of course, the suggestion could be changed if some day will be possible to declare as a primary and foreign key being component of a collection, using a kind of function like this - ONE_ELEMENT_OF:

```
CREATE TYPE book_id_type AS TABLE OF VARCHAR(15);
```

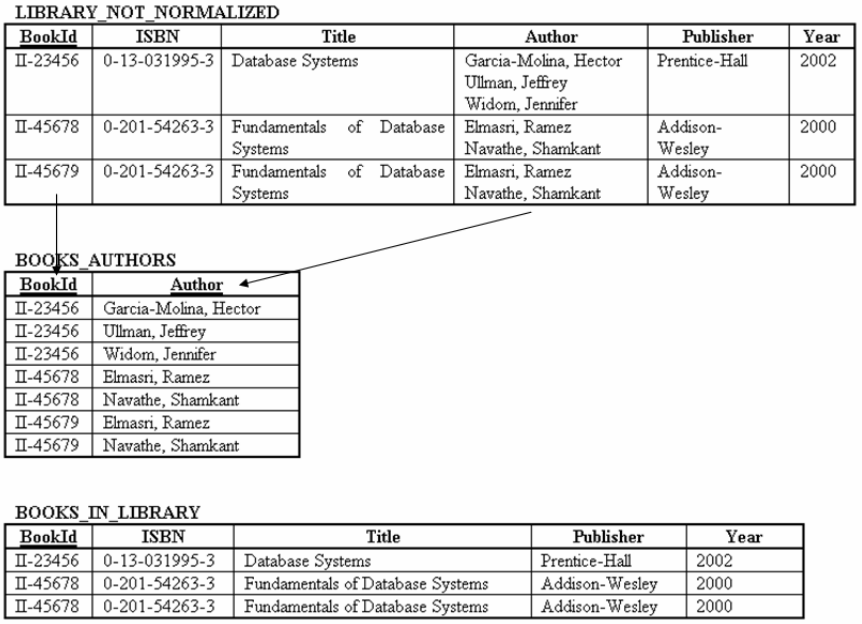
```
CREATE TABLE books (
        isbn VARCHAR(14),
        title VARCHAR(100),
        bookids BOOK_ID_TYPE,
        …,
        PRIMARY KEY ONE_ELEMENT_OF (bookids)
) ;
```

A basic rule for 1NF may be established: whenever a non-atomic (composite, collection type) attribute is to appear in a foreign key relationship, it must be “atomized”; otherwise, one can keep in its “repetitive”, collection form.

Getting Rid of Repeating Groups

When dealing with repeating groups, many authors - including Codd (1970) - recommend to remove every attribute that violates 1NF and to place it into a separate relation along with the original relation primary key (e.g. Elmasri & Navathe, 2000, pp.485-488; Connoly & Begg, 2000, p.107; Simsion, 2001, pp. 46-48).

In Figure 1, the relation LIBRARY_NOT_NORMALIZED describes the books in a library, where BookID identifies each "occurrence" of a title. The library has one copy of Garcia-Molina et al.'s book and two copies of Elmasri & Navathe's. By applying the suggested solution, in 1NF there will be two relations, BOOK_AUTHORS and BOOKS_IN_LIBRARY. The former’s primary key is (BookID, AUTHOR) which is not such a brilliant idea, because the authors are related to the title, not to each copy of the same book.



LIBRARY_NOT_NORMALIZED

BookId	ISBN	Title	Author	Publisher	Year
II-23456	0-13-031995-3	Database Systems	Garcia-Molina, Hector Ullman, Jeffrey Widom, Jennifer	Prentice-Hall	2002
II-45678	0-201-54263-3	Fundamentals of Database Systems	Elmasri, Ramez Navathe, Shamkant	Addison-Wesley	2000
II-45679	0-201-54263-3	Fundamentals of Database Systems	Elmasri, Ramez Navathe, Shamkant	Addison-Wesley	2000

BOOKS_AUTHORS

BookId	Author
II-23456	Garcia-Molina, Hector
II-23456	Ullman, Jeffrey
II-23456	Widom, Jennifer
II-45678	Elmasri, Ramez
II-45678	Navathe, Shamkant
II-45679	Elmasri, Ramez
II-45679	Navathe, Shamkant

BOOKS_IN_LIBRARY

BookId	ISBN	Title	Publisher	Year
II-23456	0-13-031995-3	Database Systems	Prentice-Hall	2002
II-45678	0-201-54263-3	Fundamentals of Database Systems	Addison-Wesley	2000
II-45678	0-201-54263-3	Fundamentals of Database Systems	Addison-Wesley	2000

Figure 1. An inappropriate schema obtained after following a well-known algorithm

Talking about repeating groups, Teorey (1999, 99-100), Simsion (2001, 44) wrongly consider relations like BOOKS (ISBN, TITLE, PUBLISHER, YEAR, AUTHOR1, AUTHOR2, AUTHOR3, AUTHOR4) as containing repeating groups. In fact BOOKS fulfills 1FN basic requirements, and none of its eight attributes is a repeating group. Nevertheless, the inappropriateness of the schema is obvious.

NULLs and the Apparently Vicious Cycle of 1NF

Normalization starts with the Universal Relation (UR) and then proceeds with elimination of partial FDs by splitting UR into new relations where all non-key attributes depend totally on the key. But if "the definition of FD is usually only applied to relations in first normal form" (Schmid & Swenson, 1975), then how to decompose URs that have repeating groups (see section above) or entity integrity constraint violations ?

A vicious cycle seems to occur in the case of URs with nulls in candidate keys, because:

- there is no 1NF of the relation, because the relational model forbids key attributes with null (meta)values – the entity integrity (Codd, 1979; Date, 1986, p.252);
- decomposition of a relation must be proceeded according only to dependencies among attributes;
- FDs are identified and analysed only after the UR fulfills 1NF requirements (Teorey, 1989; Simsion, 2001; Date, 2005).

This problem was raised by Philip (2002) who discusses the normalization of relations with null in candidate keys, typically the relations representing "information on two entities, including their identifiers, and their relationship when relationship between two entities is (zero-or-one)-to-(zero-or-one)". He suggests correcting the problem by incorporating the concept of entity integrity rule into the BCNF definition.

Apparently, practitioners could apply two solutions for dealing with UR's which break entity integrity rule. First one is to temporarily ignore the entity integrity and (shamefully) accept a relation which violates a relational model basic principle, hoping that further normal forms will eliminate the embarrassing violation. The argument is that, anyway, 1NF is not the ultimate form of the schema. The second solution consists in identifying, for composite candidate keys that could have null components, all the FDs in which the nullable components are sources, and then, based on these FDs, to proceed with splitting the UR similarly to converting UR in 2NF. When all the components of UR primary key could be nullable, bringing UR in 1NF (a 1NF which does not violates any structural constraint of relational model) will be, in the same time, equivalent to transforming UR into 2NF !

From my point of view, Philip's approach is flawed, because "traditional" 3NF and BCNF applies to relations (obeying relational model rules), and not to SQL tables. SQL has been fiercely criticized for breaking relational rule of tuple uniqueness. Consequently, an SQL table

could have nulls in key attributes values, although is it quite possible to enforce the rules through PRIMARY KEY and UNIQUE with NOT NULL options in CREATE/ALTER TABLE statements. Simply stated, a normalized relation – being necessarily at least in 1NF - is a relation where null is not allowed in key attributes. In such a relation which links two entities, the tuples referring to an entity not linked to the other entity (without corresponding entity) cannot be inserted into relation - the famous insert anomaly.

Difficulties in Capturing Relationships Among Attributes Through Dependencies

The basic ingredient of normalization is the functional dependency (FD). FD is a simple, but extremely powerful concept, linking semantically two or more attributes. An attribute (or collection of attributes) B is functionally dependent on attribute (or collection of attributes) A within a relation R if, at every instant of time, each value of A has no more than one value of B associated with it under R (Schmid & Swenson, 1975): $A \rightarrow B$.

Many attempts have struggled to automate FDs discovery among the database attributes – see papers mentioned by Matos & Grasser (2004) – few of them claiming the implementation of some algorithms using programming languages. Nevertheless, from my point of view, while FDs and MVDs catch semantic constraints, and while computers still have serious problems dealing with *meanings*, any attempt to formalize FD and MVD discovery would have minimal impact into a real-world database design project.

Besides its semantic power, FD has serious limitations, so that questioning the adequacy of FD for expressing comprehensive knowledge about the world (Schmid & Swenson, 1975) is a rhetoric exercise. Identifying FD limits could be much more productive. One such limit is the

inability of expressing the fact that an entity is associated with one ore more *specific* entities. A person, identified by PERSON_ID, has a name and a date of birth, therefore the following FDs work properly: PERSON_ID → PERSON_NAME, PERSON_ID → DATE_OF_BIRTH. But the fact that a specific person has exactly two children couldn't be expressed by FD PERSON_ID —/→ CHILD_NAME. Similarly, there are situations where a supplier is authorized to supply only certain products, a book has authors, a student has taken a number of courses, etc.

Apart from some cases where MVD theory could be applied, at least two solutions could solve such a kind of problem, both of them having a certain degree of artificiality. First one consists in adding one or more attributes in order to generate a FD for "catching" the relationship. For instance, we can use CHILD_NO, defined on the domain (1,2,… - positive integers greater than zero), which shows the order children were born for the current parent: (PERSON_ID, CHILD_NO) → CHILD_NAME. Analogously, for the library database, (ISBN, AUTHOR_NO) → AUTHOR_NAME and (SUPPLIER_ID, PRODUCT_SUPPLIED_NO) → PRODUCT_ID.

Database designers could nevertheless be tempted in following the second solution based on surrogate attributes. Each relation between a person (parent) and one of his/her children could be identified by a special attribute: PARENTAL_ID. Therefore, PARENTAL_ID → PERSON_ID, and PARENTAL_ID → CHILD_NAME. Similarly, CONTRIBUTION_ID → ISBN, CONTRIBUTION_ID → AUTHOR_NAME and SUPPLY_ID → SUPPLIER_ID, SUPPLY_ID → PRODUCT_ID.

The same solution can be applied in the case of multiple relationships described by Atzeni & Parker (1982) – JOBS–PARTS–SUPPLIERS. The facts that every job requires certain parts, a supplier can offer only certain parts, and for every job one or more suppliers supply only

certain parts, could lead to the idea of using three different attributes for identifying the relationships described:

- PARTS_REQUIRED_BY_JOBS_ID → JOB_ID, PARTS_REQUIRED_BY_JOBS_ID → PART_ID;
- SUPPLIER_PARTS_ID → SUPPLIER_ID, SUPPLIER_PARTS_ID → PART_ID;
- PARTS_SUPPLIED_FOR_A_JOB_ID → JOB_ID, PARTS_SUPPLIED_FOR_A_JOB_ID → PART_ID, PARTS_SUPPLIED_FOR_A_JOB_ID → SUPPLIER_ID.

Perhaps the most difficult issue regarding the DFs is that between two attributes could exist, within the same database, different relationships. Taking from Kent (1981) an example about the relation SK-DEP {EMPLOYEE_ID, SKILL, DEPENDENT}. Every employee has a number of skills and some dependents, so that EMPLOYEE_ID →→ SKILL | DEPENDENT. Now, the database must capture the following information: employees teach some of their skills to some of their dependents. Put into other words, every employee teaches not necessarily all his/her skills, and each dependent could learn different skills than other dependents of the same employee. Within the same three attributes there are three different relationships (predicates):

- every employee could have zero, one, or more skills;
- every employee could have zero, one, or more dependents to take care of;
- an employee could teach one or more of his/her skills to one or more of his/her dependents.

The multi-valued dependency is still valid (employee has skills/dependents), and, simultaneously, not valid (due to the way the employee teaches skills to dependents).

Ullman's answer to this problem is keeping the initial relation SK-DEP as the universal one. For representing the fact that an employee *e* has a skill *s*, but teaches it to none of his/her children, the tuple (*e*, *s*, *null*) will be necessarily included in the SK-DEP relation (Ullman, 1982).

There is also a (more tedious) solution which consists in replicating the attributes involved in multiple relationships. In our case (actually, William's), we could add SKILL_TAUGHT and DEPENDENT_TAUGHT attributes which are connected through inclusion dependencies to SKILL and DEPENDENT. The MVD is valid and the teaching role is taken by the two new attributes.

The latter solution requires employing inclusion dependencies (Fagin, 1981; Casanova & Fagin, 1982; Beeri & Korth, 1982; Mitchell, 1983) which can solve a major trouble: entities sometimes play multiple roles within a relational database. Ironically, if other less important concepts of normalization have been overemphasized in the publications, inclusion dependencies have received little attention in the database design books and guides.

Transitive Functional Dependencies Which Are Not Quite… Transitive

Converting a database into the third normal form (3NF) requires removal of transitive dependencies. A FD A → B is transitive if and only if there exists another attribute/group of attributes C, so that A → C and C → B. Schmid and Swenson (1975) proved that sometimes derivation of FDs through transitivity leads to other FDs which are not necessarily relevant or even correct. Besides the fact that their example (EMPLOYEE – MANAGER – SALARY – JOB_DESCRIPTION – JOB_NAME) suffers from a labeling problem which induces the error, in practice database designers are interested in eliminating the transitivity, not "encouraging" it.

Consequently, the transitivity problem we discuss below is another type. Let's take a relation R {PROFESSOR, OFFICE, DEPARTMENT, PHONE} recording the office (room) where a professor can be found (sometimes, in his/her spare time).

A professor is assigned to an office (room): PROFESSOR → OFFICE, being member of a department: PROFESSOR → DEPARTMENT. An office could be shared, sometimes against their will, by two or more professors (I really don't want to offend the professors, but these things could happen, at least in third-world universities where I belong to): OFFICE $-/\rightarrow$ PROFESSOR. Usually, each office belongs to a single department (for administrative purposes): OFFICE → DEPARTMENT and has a single telephone OFFICE → PHONE.

Because PROFESSOR → OFFICE → DEPARTMENT, dependency PROFESSOR → DEPARTMENT is quite transitive and, therefore, R must be split into R1 {OFFICE, DEPARTMENT, PHONE} and R2 {PROFESSOR, OFFICE}. By joining the two relations it is easy to find out which department a professor works in.

Based on some real-cases, we can imagine the following scenario: due to some space problems, for a period of time, a professor could be assigned to an office together with a colleague from another department. The office continues to belong to the same initial department, but it's shared by two professors coming from different departments. Each above FD is still valid, but answering the question "which department professor X belongs to ?" would have a fallacious result.

This type of problem has an easy solution. Instead of having one attribute, DEPARTMENT, there could be used two: DEPT_PROF (department-where-professor-belongs-to) and DEPT_OFFICE (department-taking-care-of-the-office). Now, PROFESSOR → DEPT_PROF, PROFESSOR →OFFICE, OFFICE → DEPT_OFFICE, OFFICE → PHONE, and the two relations are R1 {OFFICE, DEPT_OFFICE, PHONE} and R2 {PROFESSOR, OFFICE, DEPT_PROF}. The annoying fact that both attributes reflect the same thing (a department) - and this is clearly a redundancy -, could be fixed using inclusion dependencies.

Analogously, Atzeni & Parker's (1982) case PROJ - EMP# - XEROX# can be solved. The FDs proposed by the two authors – PROJ → EMP# (each project has one manager) and EMP# → XEROX# (each secretary has one xerox key#) based on which the transitivity PROJ → XEROX# (each project has only a xerox key number) are flawed, because a project groups many employees, but only one is the project manager. Consequently, the right dependencies are: PROJ → MGR# (a project has a manager), MGR# → EMP# (every manager is an employee, too), SECR# → XEROX# (a secretary has at most one xerox key), and also SECR# → EMP#. Actually, managers and secretaries are subtypes of employees; every manager and every secretary is an employee, but the vice-versa does not hold.

Non-Canonical Functiona Dependencies (Another Kind of Transitivity)

According to Date & Fagin (1992) a relation schema is in third normal form if whenever X→ A is a nontrivial FD of the schema, where A is a single attribute, then either X is a superkey or A is a key attribute. When working with FDs a common advice is to place only one attribute to the right side – FD's destination (Saleh, 1994, p.76; Garcia-Molina et al. 2002, p.89) - and then achieve the *canonical* FD form. Usually this is acceptable, but sometimes the canonical form could lead to problems. Relation R {ORDER_ID, ORDER_DATE, SUPPLIER_ID, INVOICE_NO, INVOICE_DATE, INVOICE_VALUE} is designed to record information about the orders a company sends to its suppliers. In return, the supplier deliver the products (services) requested in the order, and charges the company by sending an invoice. ORDER_ID is unique, but two or more suppliers could send the company invoices with the same number. Consequently, (INVOICE_NO, SUPPLIER_ID) → INVOICE_DATE and (INVOICE_NO, SUPPLIER_ID) → INVOICE_VALUE. If each invoice corresponds to one or more orders, but each order

corresponds to a single invoice, it is obvious that there are two transitive FDs: ORDER_ID → (INVOICE_NO, SUPPLIER_ID) → INVOICE_DATE and ORDER_ID → (INVOICE_NO, SUPPLIER_ID) → INVOICE_VALUE. In this case, R is not in 3NF and the decomposition is based only on non-canonical FDs.

Some Problems with BCNF Definition

1NF, 2NF, and 3NF definitions are easy to understand by any practitioner. Yet this is not the case for BCNF. Discovered by Heath, Boyce and Codd, and introduced in (Codd, 1974), BCNF deals with situation where a key attribute (component) depends on non-key attribute: relation R(A, B, C) where FDs (A, B) → C and C → B hold, is in 3NF but not in BCNF (Beeri & Bernstein, 1979). Thus, BCNF demands that the only nontrivial functional dependencies are the result of keys, whereas 3NF is less demanding, allowing nontrivial FD X → A, if A is part of a key (Date & Fagin, 1992).

Date (1986, 374), presenting BCNF, stated "...Codd's original definition of 3NF suffered from certain inadequacies. To be precise, it did not deal satisfactorily with the case of a relation that: (a) had multiple candidate keys, (b) where those candidate keys were composite, and (c) the candidate keys overlapped (i.e. had at least one attribute in common)".

Now, there are some counterexamples to Date's assertion. Considering relation PRODUCTS_ORDERED {ORDER_ID, ORDER_LINE, PRODUCT_ID, QUANTITY, PRICE_PER_UNIT}. ORDER_LINE takes the values 1, 2 ... indicating the original sequence in which products appear within each order. A product can appear only once in an order, therefore the relation: (a) has two candidate keys: (ORDER_ID, ORDER_LINE) and (ORDER_ID, PRODUCT_ID), (b) both candidate keys are composite, (c) they overlap

(ORDER_ID is common). Still, PRODUCTS_ORDERED is really in BCNF and decomposing it is not recommended, for the sake of information preservation.

Actually, Beeri & Bernstein identified some BCNF inaccuracies even in 1977 edition of Date's book (Beeri & Bernstein, 1979). Unfortunately neither their real-world example (ADDRESS, POSTAL_CODE, CITY) is clear.

The lack of appetency for BCNF practical case studies in normalization literature could be explained by its unsuitability towards designing schemata according to the representation (of semantic constraints – FDs, MVDs) principle (Zaniolo, 1982). Beeri, Bernstein, & Goodman (1978), Beeri & Bernstein (1979), Zaniolo (1982) argued that historical transition from 3NF to BCNF cannot simplistically labeled an advance, but must regarded as a step with both advantages and problems.

FOGGY, DEBATABLE, AND PURE THEORETICAL EXAMPLES

A large number of books and articles about normalization suffer from lack of relevance of their examples and case studies, and sometimes misinterpret the modeled processes and transactions. Starting with Kent (1983a) who presents a special type of redundancy - *inter-record redundancy* - using three relations: R1 {EMPLOYEE, DEPARTMENT}, R2 {DEPARTMENT, LOCATION}, and R3 {EMPLOYEE, LOCATION}, it is rather doubtful how these relations appear, because if using either UR decomposition or synthesis, the transitive dependency is obvious: EMPLOYEE → DEPARTMENT → LOCATION. R3 simply cannot result though normalization, so the example is flawed.

As expected, the most inappropriate examples and case studies concern multivalued dependencies (MVDs) and forth normal form (4NF), and also join dependencies (JDs) and fifth normal form (5NF). The problem was aknowledged by Date and Fagin (Date & Fagin, 1992).

All the practical difficulties in identifying real-world MVDs come from their definition (Fagin, 1977): X →→ Y holds for the relation R (X, Y, Z) if R is the join of its projections R1 (X, Y) and R2 (X, Z). Can somebody handle tens, maybe hundreds of attributes and thousands, or even millions of rows, discovering the situations where the above theorem holds ? Fortunately, there are some clues about the "independence" of Y and Z versus X.

Based on Fagin's paper, Kent (Kent, 1981; Kent, 1983a) and Date (Date, 1986, pp. 381-385) emphasized that MVD can be analyzed between two attributes (or groups of attributes) but only in the presence of a third attribute/group. Nevertheless, respectable contributors, such as Smith (1985), Garcia-Molina et al. (2002), ignore this request; others, like Kent (1981), present incautiously some of the examples – e.g. PERSON →→ FATHER.

Smith (1985) stated that there is a MVD from A to B, A →→ B, if, at any point in time, a fact about A determines a set of facts about B. The definition is fallacious because there are many cases when a value of A determines a set of values of B (e.g. a value of ORDER_NO determines a set of values of ORDER_ITEM, but there is no MVD between ORDER_NO and ORDER_ITEM). Also Riordan (1999, 40) commits a similar mistake. DMV inconsistencies have appeared not only in valuable database design books (e.g. Simsion, 2001, pp.245-248), but also in papers of normalization classics – see the LANDLORD-OCCUPANT–ADDRESS case - identified by Kent (1981) in (Beeri, Bernstein, & Goodman, 1978) and even in Fagin's original paper - the unexplained MVD CLASS →→ TEXT - a class of students use a set of textbooks (Fagin, 1977) – in which the TEXT's "witness" atribut/group required for a MVD lacks.

Another debatable example is taken from the paper that originated the MVDs (Fagin, 1977). Developing MVD theory and illustrating it with examples, Fagin starts with relation S* (EMPLOYEE, SALARY, YEAR, CHILD) in which there are two MVDs : EMPLOYEE →→ (SALARY, YEAR) and EMPLOYEE →→ CHILD. In order to make the example more illuminate, he introduces the attribute SPOUSE and, of course, when spouse arrives, the troubles are not far away… Rigorously speaking, R. Fagin has a problem with the spouse ! He states that not only EMPLOYEE →→ SPOUSE, but also EMPLOYEE → SPOUSE (no bigamy in the USA !) holds. The dependency is valid just for showing the *current* spouse. But the salary is recorded every year. For a past year, on a correspondent tuple, the salary value is correct, but at that time the value of the spouse (sorry for the terminology) might be different than current value (in that year the emplyee had a different spouse).

Even so, R (EMPLOYEE, SALARY, YEAR, CHILD, SPOUSE) must be decomposed before taking into account the MVDs. Each year an employee has a salary: (EMPLOYEE, YEAR) → SALARY, and currently an employee could have zero or one spouse: EMPLOYEE → SPOUSE. Obviously, R is not in 2NF and must be split into: R1 (EMPLOYEE, YEAR, SALARY), R2(EMPLOYEE, SPOUSE) and R3 (EMPLOYEE, YEAR, CHILD). If relations could be described in terms of predicates, what can be said about R3 ? At the first sight, a tuple refers to an employee who has a child born in a specific year, but it is not the case. We'll be back soon.

Coming closer to reality, in relation R (EMPLOYEE, SALARY, YEAR, CHILD, SPOUSE) an employee has a salary that could be modified each year, has one or more children, each child being made with one spouse. In Fagin's example, a child could appear in a tuple with the current spouse of the employee, even when the offspring comes from a previous marriage !

If we accept that employees would not accept having two children with the same name, and also that, during the years, every employee can have zero, one, or more marriages (with or

without resulting children), there will be other FDs: (EMPLOYEE, YEAR) → SALARY and (EMPLOYEE, CHILD) → SPOUSE. Each employee has at most one current SPOUSE, but the dependency is not EMPLOYEE → SPOUSE, because in this case the FD (EMPLOYEE, CHILD) → SPOUSE is partial. A better solution is to use two different attributes for the spouse, one for the parentship - (EMPLOYEE, CHILD) → SPOUSE_PARENT and another for recording the current spouse, EMPLOYEE → CURRENT_SPOUSE. Consequently, relation R0 (EMPLOYEE, SALARY, YEAR, CHILD, SPOUSE_PARENT, CURRENT_SPOUSE) is decomposed into R11(EMPLOYEE, YEAR, SALARY), R12 (EMPLOYEE, CHILD, SPOUSE_PARENT), R13 (EMPLOYEE, CURRENT_SPOUSE) and the remainings of R0: R14 (EMPLOYEE, YEAR, CHILD).

Now, turning back as promised, relation R3 has a similar structure to R14. Depending of the hypothesis, in these relations FDs or MVDs could hold. If we agree that YEAR designates only annual salary for an employee, the MVD EMPLOYEE →→ YEAR | CHILD will hold. If we state that YEAR designates both the year of a employee's salary and child's birth year, there will be a FD: (EMPLOYEE, CHILD) → YEAR. In either cases R3/R14 will be broken.

Another wrong illustration of MVDs is in Atzeni and Parker (1982). Wanting to prove that sometimes lossless joins may not be correct or meaningful, they discuss the following database schema: TAKE (COURSE, STUDENT), TEACH (COURSE, TEACHER), ADVISE (STUDENT, TEACHER, THESIS_TITLE). The suggested dependencies COURSE →→ STUDENT | TEACHER and (STUDENT, TEACHER) → THESIS_TITLE don't really hold because:

- if a course is taught for all students by a single teacher, then COURSE → TEACHER;
- if the same course is taught by two or more professors, then COURSE —/→→ STUDENT | TEACHER, because the student is enrolled in a single section of that course; e.g. if the course Databases is taken by student Lewitt who enrolls at the

professor Kent's section, and also if the same course is taken by student Kohl who enrolls at professor Smith's section, that means neither that student Lewitt will attend professor Smith's section of database course, nor that student Koll will attend professor Kent's section.

For many contributors, the best way to protect themselves from error is to keep discussion within a theoretical framework, not taking the risk of real-world examples (e.g. Savnik & Flach, 2000). In any case, hanging on theorems and lemmas seems to be less embarrassing than presenting flaw examples, such as Stephens and Plew's (2001, 196-197).

Despite the number of papers dedicated to JDs and 5NF, the case studies are extremely rare, due to the scarcity of real constraints to be fulfilled by a relation for being subject of JD/5NF analysis. This paper will keep the rarity of examples as the same level, because we shall not discuss at all any JD/5NF example, proper or wrong.

LACK OF ACCEPTED (GRAPHICAL) DIAGRAMMING TOOLS

It is often said that one picture is worth more than a thousand words. Database design is no exception to this rule. The E-R and UML tremendous success is related to the lisibility and expressivity of the results, mainly diagrams. Unfortunately, normalization is described and prescribed mostly with words plus Greek symbols and also theorems, which are not comfortable to the database designers.

FD diagrams – as in Date (1986), Pratt & Adamski (1991) - and graphs (Saleh, 1994) have been repeatedly proposed but have had little impact. The lack of a broader acceptance for one or another graphical tool has been a major drawback, and strange in a certain extent, because the theoretical ground for constructing schema by FDs synthesis emerged three decades ago. It

is equally true that FD diagrams are almost impossible to be handled "verbally" when dealing with tens or hundreds of attributes.

One of the first attempts to use graphs in designing relational databases was Schmid & Swenson's (1975). Trying to enrich the, at that time, recent-born relational model with some new semantic issues, the authors applied a graph formalism to represent relations among objects within a database. In a certain way, they anticipated Chen's E-R model (Chen, 1976), but their graph could represent only relationships among database objects, ant not FDs. Nevertheless, the idea of transforming each subgraph within the graph representation into a relation is valuable.

Bernstein (1976) proposed a technique to synthesize relations in 3NF from FDs, proving that the schema obtained contains a minimal number of relations. The algorithm is mentioned in almost every database book, but not too many authors describe it in depth (neither do I). Even proven to be correct, the algorithm is hard to apply – Salzberg (1986) is one of the few who put it to work -, mainly because of closure and minimal cover, which are very difficult to master when the number of attributes is large (this is the case of the vast majority of real world applications).

Ausiello, D'Atri and Sacca (1983) introduced one of the most suitable graph-theoretic approach for the representation of FDs. The nodes correspond to simple attributes - either source or destination in FDs -, and also to the group of attributes that form a compound FD source. A full arc connects FD's source and destination. Dotted arcs connect compound FD sources with each of their component attributes – see Figure 2. Graph lisibility is better due to representation of the *reduced* form of the FDs set. Remarkably, the authors give an algorithm for minimum covering of a nonredundant FD graph.

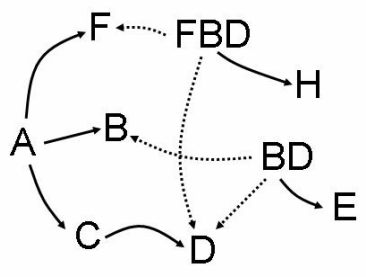


Figure 2. Ausielli, D'Atri, & Sacca 's diagramming style

The are also some major drawbacks of the graph, as follows:

- the graph includes not only FDs (full arcs), but also the connection between compounds sources and their attributes, which can complicate the figure;
- it is hard to imagine what a compound node without outgoing full arcs really means;
- neither inclusion nor multi-valued dependencies are represented.

Ausiello, D'Atri and Sacca have not developed their formalism, neither tried to apply it even to small sized applications, so that their contribution has been not acknowledged as deserves.

A famous FDs diagramming technique was elaborated by Smith (1985) who asserted that a fully normalized structure could be obtained even using only dependency diagrams, in his words, *rigorous* dependency diagrams. Diagrams proposed are actually closer to graphs than to FD diagrams presented in classical books such as Date's. Smith argues that the loss-less decomposition method is tedious to apply, and the construction of an initial UR to outset the decomposition process requires a lot of effort. Instead, he advocates for method of "simply and directly composing a myriad of unstructured data fields into fully normalized tables" (Smith, 1985).

Smith's diagrams contain bubbles in which one or more fields are included – each database field appearing, by contrast with Ausiello, D'Atri and Sacca's formalism -, only once on a diagram, and also arrows and domain flags – see Figure 3.

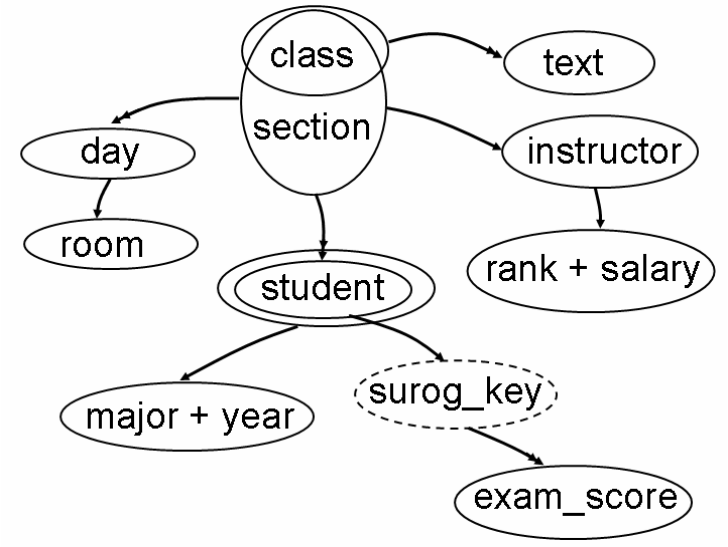


Figure 3. A Smith's diagram

Single-headed arrows linking two bubbles signalize FDs between the bubbles' attributes; double-headed arrows stand for MVDs. If FD source (determinant) has two or more destinations, then the diagram can be simplified by combining those destinations into one target bubble – as RANK + SALARY in Figure 3. Double bubbles show a linkage of a MVD destination with appears as source in some FDs and/or MVDs. Domain flags, which lack in the above diagram, are used to indicate fields sharing the same domain. A table will be composed by a FD or MVD source - along with all its destinations (attributes in the target bubbles). When a table about to be composed will have more than two or three fields in its primary key, Smith recommends using surrogate keys included in dotted bubbles. The diagram in Figure 3 uses a surrogate key for dealing with the sequence of MVDs: [CLASS + SECTION] →→ STUDENT →→ EXAM_SCORE.

Consequently, based on Figure 3 diagram, Smith composes the following tables: R221 {CLASS, TEXT}, R212 {CLASS, SECTION, INSTRUCTOR}, R211 {INSTRUCTOR, RANK, SALARY}, R11a {CLASS, SECTION, STUDENT, SUROG_KEY}, R11b {SUROG_KEY,

EXAM_SCORE}, R121 {STUDENT, MAJOR, YEAR} and R222 {CLASS, SECTION, DAY, ROOM} (Smith, 1985).

Despite the fact that some definitions - for example, MVDs - are not accurate, and the way of treating the sequence of MVDs/FDs – as in the above surrogate key case - is quite debatable, Smith's paper is highly valuable because his technique does not deal with closures, covers and other theoretical stuff, but with the real application needs. The diagrams are easy to understand, simple to apply and useful in identifying partial and transitive FDs. Last, but no least, they make the normalization work bearable even when there are tens of attributes in the database.

Smith's paper shows how to normalize a relational database without knowing and using too much theory. In addition, the article covers some more complex case studies. Sadly, Smith's methodology has not been followed up and there are no software tools to make it useful to database designers.

CONCLUSION

Despite its theoretical achievements, normalization has failed to become an ultimate guide for database designers. It is true that normalization was never claimed to be a panaceum, but the distance between theory and practice has not decreased at all in time. That is a pity because normalization has excellent sets of concepts and methodologies. In many cases, due to excessive formalization, practitioners design database structures based mostly on their intuition and experience, and, after building the logical model within E-R methodology, they try to validate their schema using normalization. Presenting normalization in relationship with E-R model could be valuable for academic purposes, but yet, normalization has almost nothing to do with

E-R modeling (there are the different methodologies based on totally different "philosophies", even they are dedicated to database logical design).

It is not our intention to seek an escape goat for normalization decay. Rather, we are interested in estimating the opportunities and risks that practitioners could experience when using normalization for real-world database design. By frankly presenting designers the weaknesses of normalization in a manner which is digestible by average people, normalization could, maybe, reduce the distance between books, research journals and day-to-day designers.

To conclude, we may say that the entire evolution of database normalization literature has proven that, despite the initial goal to serve as a logical database design methodology, normalization has mostly entertained the theorists and, with only a few exceptions, contributors and academics have failed to escape from the ivory tower of theory.

The source of the problem is that, on one hand, normalization was built as a theoretical and *rigorous* methodology, and, on the other hand, practitioners must design databases for real world applications and "real world" has proved to be *not rigorous* at all…

A previous (and reduced) form of this paper ended this way: "After all has been said and done, the main question could be: may we pronounce the divorce between normalization and database designers ? Before answering the question, it is worth thinking at another one: have they ever been married ? " [Fotache, 2005]. I still enjoy the question, so I keep it as a good final.

REFERENCES

Anthony, S.R., Batra, D. (2002). CODASYS: A Consulting Tool for Novice Database Designers, *ACM SIGMIS Database*, *33*(3), 54-68

Atzeni, P., & Parker, D.S. (1982). Assumptions in Relational Database Theory. In *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, Los Angeles, California, (pp. 1-9).

Ausiello, G., D'Atri, A., & Sacca, D. (1983). Graph Algorithms for Functional Dependency Manipulation. *Journal of the ACM*, *30* (4), 752-766

Beeri, C., Bernstein, P.A., & Goodman, N. (1978). A Sophisticate's Introduction to Database Normalization Theory. In *Proceedings of the 4th International Conference on Very Large Data Bases*, Berlin, Germany, (pp. 113-124).

Beeri, C., & Bernstein, P.A. (1979). Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Transactions on Database Systems*, *4* (1), 30-59

Beeri, C., & Korth, H.F. (1982). Compatible Attributes in a Universal Relation. In *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, Los Angeles, California, (pp. 55-61).

Bell, A.E. (2004). Death by UML Fever. *Queue*, *2* (1), 72-80

Bernstein, P.A. (1976). Synthesizing Third Normal Form Relations from Functional Dependencies. *ACM Transactions on Database Systems*, *1* (4), 277-298

Biskup, J. (1998). Achievements of Relational Database Schema Design Theory Revisited. In *Semantics in Databases* (Libkin, L., Thalheim, B., eds.), vol. 1358 of *Lecture Notes in Computer Science*, (pp.29-54). Retrieved November 6, 2005, from http://citeseer.ist.psu.edu/biskup98achievement.html

Biskup, J., Dayal, U., & Bernstein, P.A. (1979). Synthesizing Independent Database Schemas. In *Proceedings of ACM SIGMOD Conference on Management of data*, Boston, Massachusetts (pp.143-151).

Camps, R. (1996). Domains, Relations and Religious Wars. *ACM SIGMOD RECORD, 25* (3), 3-9.

Carlis, J., & Maguire, J. (2001). *Mastering Data Modeling. A User-Driven Approach*, Boston, Massachusetts: Addison-Wesley.

Casanova, M.A., Fagin, R., & Papadimitriou, C.H. (1982). Inclusion Dependencies and Their Interaction with Functional Dependencies *(Extended abstract)*. In *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, Los Angeles, California, (pp. 171-176).

Chen, P. (1976). The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, *1* (1), 9-36

Codd, E.F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, *13*(6), 377-387.

Codd, E.F. (1972). Further Normalization of the Data Base Relational Model. In *DataBase Systems*, *Courant Computer Science Symposia Series*, *6*, Englewood Cliffs, New Jersey: Prentice-Hall, (pp. 33-64).

Codd, E.F. (1974). Recent Investigations into Relational Database Systems. In *Information Processing 74*, North-Holland: Amsterdam, Holland, (pp. 1017-1021).

Codd, E.F. (1979). Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, *4* (4), 397-434

Conception, A.I., & Villafuerte, R.M. (1990). ExpertDB: An Assistant Database Design System. In *Proceedings of the 3rd international conference on Industrial and engineering applications of artificial intelligence and expert systems (vol.1)*, Charleston, South Carolina, (pp. 333-339).

Connolly, T., & Begg, C. (2000). *Database Solutions. A step-by-step guide to building databases*, Harlow, England: Addison-Wesley.

Connolly, T., & Begg, C. (2002). *Database Systems. A practical Approach to Design, Implementation and Management* (3rd edition), Reading, Massachusetts: Addison-Wesley.

Darwen, H., & Date, C.J. (1995). The Third manifesto. *ACM SIGMOD RECORD, 24* (1), 39-49.

Date, C.J. (1986). A*n Introduction to Database Systems* (4th edition). Reading, Massachusetts: Addison-Wesley.

Date, C.J. (2003). *What First Normal Form Really Means*. Retrieved on November, 12, 2004 from *http:/www.dbdebunk.com*

Date, C.J. (2004). *An Introduction to Database Systems* (8th edition). Boston, Massachusetts: Addison-Wesley.

Date, C.J. (2005). *Database in Depth. Relational Theory for Practitioners*. Sebastopol, California: O'Reilly.

Date, C.J., & Fagin, R. (1992). Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases. *ACM Transactions on Database Systems*, *17* (3), 465-476

Diederich, J., & Milton, J. (1988). New Methods and Fast Algorithms for Database Normalization. *ACM Transactions on Database Systems*, *13* (3), 339-365

Elmasri, R., & Navathe, S.R. (2000). *Fundamentals of Database Systems*. Reading, Massachusetts: Addison-Wesley.

Erickson, J., Lyytinen, H., Siau, K., (2005). Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research, *Journal of Database Management*, *16*(4), 88-100

Fagin, R. (1977). Multivalued Dependencies and a New Normal Form for Relational Databases. *ACM Transactions on Database Systems*, *2* (3), 262-278

Fagin, R. (1979). Normal Forms and Relational Database Operators. In *Proceedings of ACM SIGMOD Conference on Management of data*, Boston, Massachusetts, (pp. 153-160).

Fagin, R. (1981). A Normal Form for Relational Databases That Is Based on Domains and Keys. *ACM Transactions on Database Systems*, *6* (3), 387-415

Fagin, R., Mendelzon, A.O., & Ullman, J.D. (1982), A Simplified Universal relation Assumption and Its Properties, *ACM Transactions on Database Systems*, 7 (3), 343-360

Fleming, C., & von Halle , B. (1989). *Handbook of Relational Database Design*. Reading, Massachusetts: Addison-Wesley.

Fotache, M. (2005). Database Designers and Normalization: Anatomy of a Divorce. In *Proceedings of the 8th International Conference on Business Information Systems*, Poznan, Poland, (pp. 330-341).

Garcia-Molina, H., Ullman, J., & Widom, J. (2002). *Database Systems. The Complete Book*. Upper Saddle River, New Jersey: Prentice Hall.

Hainaut, J.L., Cadelli, M., Decuyper, B., Marchand, O. (1992). Database Case Tool Architecture: Principles For Flexible Design Strategies. In *Proceedings of the fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, Manchester, United Kingdom (pp.187-207).

Halpin, T. (2001). *Information Modeling and Relational Databases*. San Francisco, California: Morgan Kaufmann.

Harrington, J. (2002). *Relational Database Design Clearly Explained* (2nd edition). Amsterdam, Holland: Morgan Kaufmann.

Hay, D.C. (1999). Object Orientation and Information Engineering: UML. *The Data Administrator Newsletter*, *9*. Retrieved on November 22, 2005 from *http://www.tdan.com/i008fe02.htm*

Hernandez, M. (1997). *Database Design for Mere Mortals. A Hands-On Guide to Relational Database Design*. Boston, Massachusetts: Addison-Wesley.

Ilyas, I.F., Markl, V., Haas, P., Brown, P., & Aboulnaga, A. (2004). CORDS. Automatic Discovery of Correlations and Soft Functional Dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, Paris, France (pp.647-658).

Jaeschke, G., & Schek, H.J. (1982). Remarks on the Algebra of Non First Normal Form Relations. In *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, Los Angeles, California, (pp. 124-138).

Kent, W. (1981). Consequences of Assuming a Universal Relation. *ACM Transactions on Database Systems*, *6* (4), 539-556

Kent, W. (1983a). A Simple Guide to Five Normal Forms in Relational Database Theory. *Communications of the ACM*, *26* (2), 120-125

Kent, W. (1983b). The Universal Relation Revisited. *ACM Transactions on Database Systems*, *8* (4), 644-648

Levene, M., & Loizou, G. (1991). A Domain Theoretic Characterization of the Universal Relation. *International Journal of Computer Mathematics*, *40* (142). Retrieved on November 6, 2005 from *http://citeseer.ist.psu.edu/levene91domain.html*

Levene, M., & Vincent, M.W. (2000). Justification for Inclusion Dependency Normal Form, *IEEE Transactions on Knowledge and Data Engineering*, *12*(2), 281-291

Lien, E. (1981). Hierarchical Schemata for Relational Databases. *ACM Transactions on Database Systems*, *6* (1), 48-69

Ling, T.W., & Goh, C.H. (1992). Logical Database Design with Inclusion Dependencies. In *Proceedings of the Eighth International Conference on Data Engineering*, Tempe, Arizona (pp. 642-649).

Ling, T.W., Tompa, F.W., Kameda, T. (1981). An Improved Third Normal Form for Relational Databases. *ACM Transactions on Database Systems*, *6* (2), 328-346

Maier, D., Ullman, J.D., Vardi, M.Y. (1984). On the Foundations of the Universal Relation Model. *ACM Transactions on Database Systems*, *9* (2), 283-308

Makinouchi, A. (1977). A Consideration on Normal Form of Not-Necessarily-Normalized Relations in the Relational Data Model. In *Proceedings of the 3rd International Conference on Very Large Data Bases*, Tokyo, (pp.447-453)

Mannila, H., & Raiha, K.J. (1986). Inclusion Dependencies in Database Design. In *Proceedings of the 2nd International Conference on Data Engineering*, Los Angeles, California, (pp. 713-718).

Markowitz, V.M., & Shoshani, A. (1992). Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach, *ACM Transactions on Database Systems*, *17*(3), 423-464

Matos, V., & Grasser, B. (2004). SQL-based Discovery of Exact and Approximative Functional Dependencies. *INROADS – The SIGCSE Bulletin*, *36* (4), 58-63

Mitchell, J., (1983). Inference Rules for Functional and Inclusion Dependencies. In *Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, Atlanta, Georgia, (pp. 58-69).

Muller, R. (1999). *Database Design for Smarties. Using UML for Data Modeling*. San Francisco, California: Morgan Kaufmann

O'Neil, P., & O'Neil, E. (2001). *Database. Principles, Programming, Performance*. San Francisco, California: Morgan Kaufmann.

Osborn, S.L., & Heaven, T.E. (1986). The Design of a Relational Database System with Abstract Data Types for Domains. *ACM Transactions on Database Systems*, *11*(3), 357-373

Ozsoyoglu, Z.M., & Yuan, L.Y. (1987). A New Normal Form for Nested Relations. *ACM Transactions on Database Systems*, *12*(1), 111-136

Pascal, F. (2003). *What First Normal Form Really Means Not*. Retrieved November 24, 2004 from *http:/www.dbdebunk.com*

Philip, G.C. (2002). Normalization of Relations with Nulls in Candidate Keys, *Journal of Database Management*, *13*(3), 35-45

Pratt, P.J. (1985). A Relational Approach to Database Design. In *Proceedings of the 16th ACM SIGCSE technical symposium on Computer Science education*, New Orleans, Louisiana, (pp. 184-200).

Pratt, P.J, & Adamski, J.J. (1991). *Database Systems. Management and Design* (2nd edition). Boston, Massachusetts: Boyd & Fraser.

Riordan, R. (1999). *Designing Relational Database Systems*. Redmond, Washington: Microsoft Press

Rosenthal, A., Reiner, D. (1994). Tools and Transformation – Rigorous and Otherwise – for Practical Database Design. *ACM Transactions on Database Systems*, *19*(2), 167-211

Saleh, I. (1994). *Les bases de données relationneles*. *Conception et réalisation*. Paris, France: Hermes.

Salzberg, B. (1986). Third Normal Form Made Easy. *ACM SIGMOD RECORD, 15* (4), 2-18.

Savnik, I., & Flach, P. (2000). *Discovery of Multivalued Dependencies from Relations*, Retrieved on December 5, 2004 from http://www.informatik.uni-freiburg.de/~dbis/Publications/2k/report00135.ps.gz

Schmid, H.A., & Swenson, J.T. (1975). On the Semantics of the Relational Data Model. In *Proceedings of the 1975 ACM SIGMOD international conference on Management of data*, San Jose, California, (pp. 211-223).

Sciore, E., (1983). Inclusion Dependencies and the Universal Instance. In *Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, Atlanta, Georgia, (pp. 48-57).

Siau, K., Erickson, J., Lee, L.Y. (2005). Theoretical vs. Practical Complexity: The Case of UML, *Journal of Database Management*, *16*(3), 40-57

Simsion G.C. (2001). *Data Modeling Essentials*.(2nd edition). Scottsdale, Arizona: Coriolis.

Smith, H.C. (1985). Database Design: Composing Fully Normalized Tables From a Rigorous Dependency Diagram, *Communication of the ACM*, 28 (8), 826-838

Stephens, R.K., Plew, R.R. (2001). *Database Design*. Indianapolis, Indiana: Sams.

Teorey, T. (1999). *Database Modeling & Design*. San Francisco, California: Morgan Kaufmann.

Teorey, T., Yang, D., & Fry, J. (1986). A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model, *Computing Surveys*, 18 (2), 197-222

Ullman, J. (1982). The U.R. Strikes Back. In *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, Los Angeles, California, (pp. 10-22).

Vincent, M.W. (1998). Redundancy Elimination and a New Normal Form for Relational Database Design. In *Semantics in Databases* (Libkin, L., Thalheim, B., eds.), vol. 1358 of *Lecture Notes in Computer Science*, (pp.247-264). Retrieved November 4, 2005, from *http://citeseer.ist.psu. edu/vincent98redundancy.html*

Wu, M.S. (1992), The Practical Need for Fourth Normal Form. In *Proceedings of the 23rd ACM SIGCSE technical symposium on Computer Science education*, Kansas City, Missouri, (pp. 19-23)

Zaniolo, C. (1982). A New Normal Form for the Design of Relational Database Schemata, *ACM Transactions on Database Systems*, *7*(3), 489-499

Zaniolo, C., & Melkanoff, M. (1981). On the Design of Relational Database Schemata, *ACM Transactions on Database Systems*, *6*(1), 1-47