LIBERTAS MATHEMATICA, VOL.11 (1991)

THE PROCESS OF NORMALIZATION FOR VERY COMPLEX SOFTWARE DESIGN SYSTEMS.

Viorel Vlad

The scope of this article is to present a short correlation between the "theory of normalization" and its practical implementation in the production data field for large application software development projects.

The use of good structured development methods is needed to help management have confidence that information systems will be available on time and provide the accurate management information needed to make more informed decisions.

An application software developer always wants to create a design that faithfully models a "real-word" situation. It always will try to map the conceptual model of reality onto its representation in the data base. Whatever application database package you use to support your design (Oracle, Informix, Unify, Accell, Focus etc) you are always faced with the same type of questions related to the storage of the information and what are the relationships between some groups of information.
A data analysis dilemma arises and the way you approach this will contribute to the success or failure of your design. For large, complex multi user systems a formal analysis and specific design criteria are (and must be) required.
There is no shortage of literature and opinions on how to analyze data, how to design a database, and what CASE tool to use for an accurate and expeditive design.
From my own experience, I realize that all of them are very consistent, the only difference being the type of language used to describe entities and attributes and the approach's point of view.
A big question is always: "Should we use normalization techniques for a better layout of data or not?" The answer always must be "yes", the only caveat being to what degree of expectancy you want your data normalized. One of the biggest drawbacks today in the real word of the application developer is that a distinction is not made between the Logical View Of Data and the Physical View Of Data. AT&T methodology involves producing two distinct analyses and design of data:

A. Logical View Of Data,

B. Physical View Of Data.

Always separate the physical from the logical to be able to assure data independence. The suggested basic steps in designing a standard customer application design are:

1. Establish a list of all the elements (entities - attributes) that you want to store in the data base;
2. Use "Normalization" techniques to normalize data;
3. Use the graphical representation of the data base (linkage diagrams and /or data flow diagrams) using the EXCELERATOR, ORACLE CASE tool or any other commercial package;
4. Determine the best way to store, access and update the tables.

Regardless of what Data Base you use the first three steps represent the Logical Data Base Design and must be done by experienced systems analysts. The last step represents the Physical Data Base Design and is a programmer's job. It is even recommended that in the first three phases, old programmers with a heavy background in traditional programming not be used. The experience has prove that they are not an asset for relational design, primarily due to ingrained procedural thinking, which should be avoided.

After all sets of tables have been generated with a collection of fields, we must ask ourselves: "Are they well defined? Can we add, delete and modify them without causing undesirable effects?"

The theory called "NORMALIZATION" was developed in the early 1970s by dr E.F. Codd to address precisely all these problems. The procedure is based on set theory and relational calculus. Before the process of normalization starts the application software developer must be aware of any inconsistencies which can occur. All data element lists (tables) must be reviewed and all inconsistencies (such as synonyms, homonyms, mutually exclusive data elements, derivable data elements) must be cleaned up.

The process of normalization involves examining the entities and certain relationships defined by the application developer to make sure that the attributes of each describe the entity itself and not some other attribute of the entity. There exist today a number of degrees of normalization, each protecting against greater degrees of data integrity problems, called in increasing order of simplicity, first normal form, second normal form....etc. Most application developers are concerned with only the first three.

FIRST NORMAL FORM (1NF)

A normalized relation is one for which each underlying domain contains atomic (nondecomposable) values only.

A record or relation is said to be in the "first normal form" when all the fields describes the key of the record or in a simple way have "no repeating groups".

This definition merely states that any normalized relation is in 1NF.

Before you choose a primary key for a specific table try to follow the following test rules:

- Determine the candidate key for your table, and any data elements to be removed. Is the remaining key still be unique?
- Is there any foreseeable situation in which this candidate key would not be unique?
- Does any part of the candidate key have undefined values? Of the remaining candidate keys, which have the fewest data elements?

So, because Relational Data Bases do not allow tables that contain repeating groups, remove the repeating data elements.

Consider the classical example of a university teaching schedule data base where course#, date, campus are the main fields in a data base table called course_sched. If our primary key is course# then date and campus are repeating groups which must be removed. Very often the situation occurs that a course is given at the same time inside the same campus or different campuses. Another example:

Field Name	Type
ckt_id	mdf_rack_id
PLNA440986	05-299, 05-599, 05-899

This table is not in 1NF because mdf_rack_id (05) is a repeating group. The only solution to put this table in 1NF form is by rearinging as follow:

Field Name	Type
ckt_id	mdf_rack_id
PLNA440986	05-299
PLNA440986	05-599
PLNA440986	05-899

Now in the table for each field you have exactly one value. Once the table is in this form, the minimum requirement has been met.

SECOND NORMAL FORM

A normalized relation is in the second normal form if all "non-key" fields are fully functionally dependent on the primary key. In other words a record is in the second normal form if it is in the first normal form AND all the fields in the record describe the entire key of the record. The basic rule to get a second normal form is:

- Always eliminate the partial functional dependence.

Example: BOOK_ORDER: (customer_name,order_date, code_number, title, author, quantity, price).

To get BOOK_ORDER into 2NF you must get rid of the partial functional dependence. This can be done by designing 2 different tables: ORDER:(customer_name, order_date, code_number, quantity), BOOK :(code_number, title, author,price).

THIRD NORMAL FORM (3NF)

A normalized relation is in the third normal form if:

- All the nonkey domains are fully functionally dependent on the primary key and,
- No nonkey domain is functionally dependent on any other nonkey domain.

In other words, the relation is in 3NF if whenever $X \longrightarrow A$ holds in R and A is not in X, then either X is a superkey for R, or A is prime.

If $X \longrightarrow A$ violests the 3NF, then either:

1. is a proper subset of a key ($X \longrightarrow A$ is a partial dependency) or
2. is a proper subset of nonkey ($X \longrightarrow A$ is a transitive dependency).

3NF eliminates all the transitive dependencies among non-identifier data elements. By transitive dependence we understand the following: If X, Y and Z are data elements, and if Y is dependent on X , and Z is dependent on Y, then data element Z is said to be transitively dependent on X via Y.

IF $X \longrightarrow Y$ and $Y \longrightarrow Z$
THEN
$Z \longrightarrow X$
(is transitively dependent on X)

Let's recap some of the important notions:

- (1NF) A table is in 1NF if and only if all repeating groups are removed. The table in any relational data base MUST be minimum in 1NF
- (2NF) A table is in 2NF if and only if it is in 1NF and every non-key field is fully dependent on the primary key
- (3NF) A table is in 3NF if and only if it is in 2NF and every non-key field is nontransitively dependent upon the primary key.

LOGICAL STEPS IN PRODUCING A NORMALIZED DATA BASE DESIGN (C Gane, T Sarson-Methodology)

Suppose that the production data are in an unnormalized relation (data structure with repeating groups). The first step before even starting the normalization process is to split the relation into one or more relations without repeating and assign one or more data elements as the primary key.

Normalized relation in **FIRST NORMAL FORM** (without repeating groups)

A. For relations whose keys have more than one domain, verify that each non-key domain is functionally dependent on the whole key, and not on just part of it. Split the relation, if necessary, to achieve this.

SECOND NORMAL FORM (all non-key domains fully functionally dependent on the primary key)

B. Verify that all non-key domains are mutually independent of each other. Remove redundant domains or split the relation as necessary to achieve this.

THIRD NORMAL FORM (all non-key domains are fully functionally dependent on the primary key, and independent of each other)

Normalization's strengths lie in defining a set of rules for the proper placement or grouping of data. It is recommended by most professionals in the field and has been used with the great success in many projects. I am an adept of this theory and I recommend that no project be done without taking this into consideration.

REFERENCES

[1] C.J. Date, An Introduction to Database Systems. Addison Wesley, 1986.

[2] C Gane & T. Sarson, Structured Systems Analysis. Prentice - Hall Inc.

[3] J.D. Ullman, Principles of Database Systems. Computer Science Press.

[4] P.P Chen, The entity-relational model: toward a unified view of data, ACM Transactions on Database Systems.

[5] R. Carnahan, An Introduction to Normalization, Database Application Development.