Overview of Normalization

By R. E. Wyllys @ University of Texas, Austin

Introduction

One of the more complicated topics in the area of database management is the process of normalizing the tables in a relational database. These notes are intended to provide you with an overview of this topic, which I hope will be helpful to you after you have gained some familiarity with the ideas of, and techniques used in, normalization.

The underlying ideas in normalization are simple enough. Through normalization we want to design for our relational database a set of files that (1) contain all the data necessary for the purposes that the database is to serve, (2) have as little redundancy as possible, (3) accommodate multiple values for types of data that require them, (4) permit efficient updates of the data in the database, and (5) avoid the danger of losing data unknowingly.

The primary reason for normalizing databases to at least the level of the 3rd Normal Form (the levels are explained below) is that normalization is a potent weapon against the possible corruption of databases stemming from what are called "insertion anomalies," "deletion anomalies," and "update anomalies." These types of error can creep into databases that are insufficiently normalized.

An "insertion anomaly" is a failure to place information about a new database entry into all the places in the database where information about that new entry needs to be stored. In a properly normalized database, information about a new entry needs to be inserted into only one place in the database; in an inadequately normalized database, information about a new entry may need to be inserted into more than one place, and, human fallibility being what it is, some of the needed additional insertions may be missed.

A "deletion anomaly" is a failure to remove information about an existing database entry when it is time to remove that entry. In a properly normalized database, information about an old, to-be-gotten-rid-of entry needs to be deleted from only one place in the database; in an inadequately normalized database, information about that old entry may need to be deleted from more than one place, and, human fallibility being what it is, some of the needed additional deletions may be missed.

An update of a database involves modifications that may be additions, deletions, or both. Thus "update anomalies" can be either of the kinds of anomalies discussed above.

All three kinds of anomalies are highly undesirable, since their occurrence constitutes corruption of the database. Properly normalized databases are much less susceptible to corruption than are unnormalized databases.

Normalization can be viewed as a series of steps (i.e., levels) designed, one after another, to deal with ways in which tables can be "too complicated for their own good". The purpose of normalization is to reduce the chances for anomalies to occur in a database. The definitions of the various levels of normalization illustrate complications to be eliminated in order to reduce the chances of anomalies. At all levels and in every case of a table with a complication, the resolution of the problem turns out to be the establishment of two or more simpler tables which, as a group, contain the same information as the original table but which, because of their simpler individual structures, lack the complication.

Single-Theme Tables

In practice, accomplishing normalization is often fairly simple. Confining the entries in any one table to data related to a single set of related attributes—what I like to call "single-theme tables"—will usually do the job. By a "single-theme table" I mean a table that concentrates either on one **concept** (i.e., typically, one entity) in the situation or on one **relationship** in the situation. The examples later in this lesson concern a hypothetical discussion of how to set up a database dealing with puppies, kennels, and tricks performed by the puppies. In terms of these examples, you will see single-theme tables dealing with one concept, (e.g., with just puppies and their names, with just tricks and the names of tricks) and with one relationship (e.g., pairings of puppies and tricks). You will also see some tables that are not single-theme tables, and you will see some of the problems that ensue from their failing to concentrate on a single theme.

If, instead of using the single-theme approach, you set out to normalize the tables in a database via a definitional approach (i.e., carefully examining tables in terms of the definitions of the various levels of normal forms), you may encounter some difficulty, or at least some tedium, in achieving complete surety that you have achieved a high level of normalization. The formal rules that follow provide a summary of the normalization process, but you will need to study them carefully and to work through several examples before you can start to feel comfortable in your understanding of normalization. Here are the formal rules of normalization, presented primarily for reference and as a summary, useful after you have learned about normalization through more detailed discussions elsewhere.

Note: The examples below (in the section entitled "An Outline of Normalization by Marc Rettig," following the section on "Formal Definitions of the Normal Forms") are intended to present a simplified introduction to the ideas of, and arguments for, normalization of databases. For a more detailed discussion of normalization, one source is my lesson entitled Steps in Normalization.

Formal Definitions of the Normal Forms

1st Normal Form (1NF)

Def: A table (relation) is in 1NF if

1. There are no duplicated rows in the table.

2. Each cell is single-valued (i.e., there are no repeating groups or arrays).

3. Entries in a column (attribute, field) are of the same kind.

Note: The order of the rows is immaterial; the order of the columns is immaterial.

Note: The requirement that there be no duplicated rows in the table means that the table has a key (although the key might be made up of more than one column—even, possibly, of all the columns).

2nd Normal Form (2NF)

Def: A table is in 2NF if it is in 1NF and if all non-key attributes are dependent on all of the key.

Note: Since a partial dependency occurs when a non-key attribute is dependent on only a part of the (composite) key, the definition of 2NF is sometimes phrased as, "A table is in 2NF if it is in 1NF and if it has no partial dependencies."

3rd Normal Form (3NF)

Def: A table is in 3NF if it is in 2NF and if it has no transitive dependencies.

Boyce-Codd Normal Form (BCNF)

Def: A table is in BCNF if it is in 3NF and if every determinant is a candidate key.

4th Normal Form (4NF)

Def: A table is in 4NF if it is in BCNF and if it has no multi-valued dependencies.

5th Normal Form (5NF)

Def: A table is in 5NF, also called "Projection-Join Normal Form" (PJNF), if it is in 4NF and if every join dependency in the table is a consequence of the candidate keys of the table.