

---

## Core Value 6 — Project Music

---

I take some pride in my ability to carry a tune, and my sense of tone is very good. Unfortunately, I do not have a sense of rhythm. I never hear the beat. Clearly, I can tell when a piano is out of tune, but I am a complete failure at dancing. My wife always asks me, "Don't you feel the music?" I don't. I just hear it. I have accepted the fact that I am rhythm challenged and will never be a successful musician. Good music, of all types, depends on both the melody and the beat successfully setting the overall mood of the piece.

In successful projects both the melody and the beat are necessary. The melody consists of the functional design that defines what the system will do, what it will look like, and what functions will be supported. The beat defines the underlying structure of the system, or how it will function. Just like a beautiful melody will fail as a musical piece without a proper beat, a set of colorful GUI screens will fail as an application without a strong underlying design. In music, composers often select the tempo of a piece prior to defining the melody. In our business, all too often, the look and feel of the screens are defined prior to defining the data. The single most important design element in any large-scale implementation effort is a solid relational design.

Many of the tools in use today are "object like", supporting concepts from Object Oriented Analysis and Design methodologies. These methodologies are a useful addition to every developer's bag of tricks; however, they do not provide a complete design environment. For most significant systems, a relational database is used to store the application data. At Metamor, we are currently building a 100-site system using a Sybase NLM, and another project is deploying 800 users on a large Ingres/VAX database. Other Metamor projects are using Oracle, Informix, and SQLserver/NT. Although the projects are widely disparate, all these projects have a common characteristic - - they use a relational database. The GUI design will fail without a strong underlying relational database design. The database design provides the rhythm or the beat of the project.

Relational database design may appear complex at first. We are frequently taught to design databases in "3<sup>rd</sup> normal form" to eliminate transient relationships and to denormalize for performance. All of these rules, properly applied, lead to better systems. Unfortunately, many capable application developers do not really understand the theoretical process of data normalization. I would like to propose a simple and straightforward set of steps for designing logical databases that restate the data normalization process in easy-to-understand language.

First, when building a new application, make a list of all data elements you need to input, output, or calculate in your system. In the database world, these fields are called attributes and are grouped into tables. In your list, every field should occur a single time even if you believe that it belongs in multiple tables. This list of data elements will be used to enforce my first rule of database design: Each field in the database should be stored in one and only one table unless the field is used as a key to relate multiple tables.

For example, in a personnel database, an employee's address may be stored in a single table. However, an employee's id may be used to relate an employee to a particular department by including the employee id as a key in both the employee and department tables. Following this example further, there is no need to repeat the employee's address in the department table because all that is needed is the employee id which can be used to reference the employee address in the employee table.

Rule number two: Repeat information by using multiple rows (records) and not multiple columns.

For example, if an employee can have multiple phone numbers, it might at first seem reasonable to include these numbers as phone\_1 and phone\_2 in the employee table. This would be a poor relational design as it limits an employee to at most two phone numbers. If only ten percent of the employees have multiple phone numbers, the space allocated to the second phone number will be wasted in ninety percent of the rows (records). A better approach to this scenario is to have a phone number table that includes phone numbers and employee ids. To find all of the phone numbers for an employee, look up those records in the phone number table for a given employee id.

Rule number three: Select the primary key in each table based on the key's ability to uniquely identify each row (record) in the table.

This rule can be used to help ensure that information is clustered together correctly into tables. For example, assume the following: (1) the employee id number has been selected as the primary key for the employee table, and (2) one of the fields in the employee table is a department name field defined as a thirty character string. Applying rule number three: does the employee id uniquely identify the department name stored in each employee record? The answer is no because the same department name is stored for many employees. An employee id or even an employee name does not uniquely identify the department name; it is identified by a department id. Therefore, consistent with rule number one (store it once unless it is used as a key), replace the department name field in the employee record with a department id field. Use the id field as a key to reference the department names table because, rather than storing a thirty character string for each employee, storing a one or two character string creates a more efficient design. In addition, if a department name needs to be changed, it can be changed in a single place rather than having to find and change the department name in hundreds or thousands of employee records.

The above rules are simple and appropriate for introducing relational database design to new developers. However, they are not complete enough for databases with hundreds of tables. In all cases, relational database design should be represented in both words and pictures. At Metamor, we are still sold on entity-relationship diagrams (ERDs) as the best way to represent a logical relational model.

In conclusion, the music of most projects today needs to include both melody and rhythm. The melody is documented in functional and detailed design specifications. It is developed using your favorite design methodology. Likewise, the beat of the project, which holds the music together, is a solid relational database design.