Design a Better Database with a Data Dictionary

Get step-by-step instructions to create a data dictionary that makes designing a database easier.

By Thomas Francl

Data dictionaries help you design databases consistently and satisfy system documentation requirements. Even though data dictionaries have been around for at least 30 years, most systems designers/programmers have shunned this technology since dictionaries are perceived as a documentation task better left to clerical staff. And they're missing out.

Creating the dictionary during the early systems design phase enhances design activities by identifying individual data elements and where they are stored in the system. This organized approach saves time and eliminates boring programming tasks. Afterward, system maintenance is easier because the dictionary forces discipline of a single definition for each data entity. Further, the clerical task of documenting the database structure is automatic.

Many large, successful software companies are among those who have shunned a data dictionary standard. For example, one high-end accounting system uses the variable name IDCUST to identify the customer number in the customer table, but then uses the variable CUSTOMER to identify the customer number in the invoice table. Imagine the confusion of a new maintenance programmer or the programmer who is contracted to create supplemental reports from this data. That company has made its workflow less efficient.

A methodology

So just what is a data dictionary and why have I learned to love them? System developers use a variety of tools when designing databases: pencil and paper, word processing, Microsoft Excel, or copying databases from similar systems. But none of these choices give you the data definition consistency forced by a data dictionary.

Database management systems such as Visual FoxPro and Microsoft SQL use data dictionaries to define the basic organization of a database. A data dictionary contains a list of all files in the database and the names and types of each field. Data dictionaries don't contain any data from the database, only bookkeeping information for managing it.

In the FoxPro environment, you create a table structure with MODIFY STRUCTURE, then build a database container consisting of those tables. Similarly, you add tables to a SQL database, and fields to each table. In these systems, tables and data fields are dependent upon one another, which means a field such as CITY could be 25 bytes long in the Customer table and only 20 bytes long in the Vendor table. Likewise, you could define a BALANCE field as a Numeric 12,2 in the customer table but as a Currency field in the Vendor table. This inconsistency causes confusion for designers, programmers, and users alike.

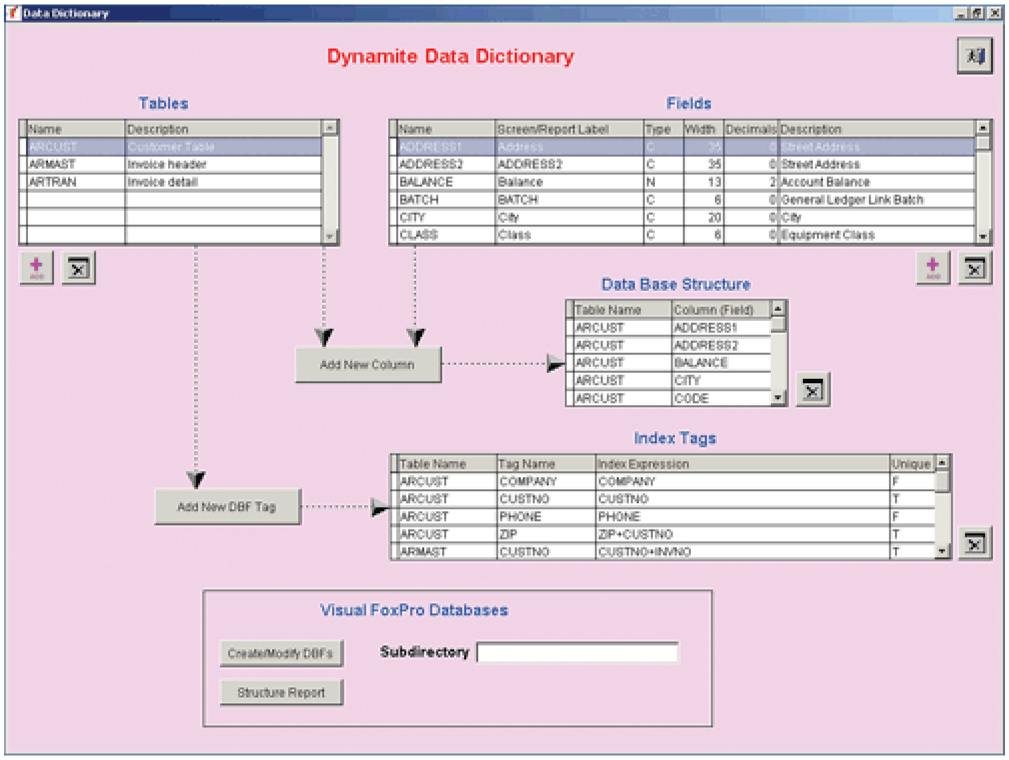
But when you use a data dictionary as a methodology — a way of creating and implementing a data dictionary as a design structure — you force consistency. As a methodology, a data dictionary manages a list of database tables and data fields (and their attributes), then specifies what fields to include in each table. This methodology could use index cards, as many programmers did in the old mainframe days, but is more commonly stored in small databases on a PC.

When I create my data dictionary, I define and maintain tables and fields independently. The table list might include Customer, Vendor, and Invoice tables. That Customer table is a Visual FoxPro DBF that contains mostly static information about the customer. Period. There is no mention of what that data is.

The list of data fields describes the individual pieces of information I'll store in the various tables. Examples include Customer Number, Company Name, Balance Owed, etc. In a FoxPro environment, you have to know that the customer number is a 10-byte character field while the balance is a 15-byte numeric field with 2 decimals. No matter what table the Customer Number is

in (Customer, Invoice, etc.), it will always have a Character-10 format.
The third table in my dictionary itemizes which data fields are in each table but makes no mention of the attributes of either the fields or the tables. The list simply includes entries such as: "CUSTOMER, CUSTNUMB", "CUSTOMER,COMPANY", "VENDOR,COMPANY", etc.

The discipline forced by your data dictionary makes screen and report design much easier, especially in a team environment. Everyone knows the attributes of every field in every table in the system. Further, should the need arise to change the field attributes, such as increasing the length of the customer number to 12 characters, you only have to modify a single data dictionary field definition.



Tables

Name	Description
ARCUST	Customer Table
ARMAST	Invoice header
ARTRAN	Invoice detail

Fields

Name	Screen/Report Label	Type	Width	Decimals	Description
ADDRESS1	Address	C	35	0	Street Address
ADDRESS2	ADDRESS2	C	35	0	Street Address
BALANCE	Balance	N	13	2	Account Balance
BATCH	BATCH	C	6	0	General Ledger Link Batch
CITY	City	C	20	0	City
CLASS	Class	C	6	0	Equipment Class

Data Base Structure

Table Name	Column (Field)
ARCUST	ADDRESS1
ARCUST	ADDRESS2
ARCUST	BALANCE
ARCUST	CITY
ARCUST	CODE

Index Tags

Table Name	Tag Name	Index Expression	Unique
ARCUST	COMPANY	COMPANY	F
ARCUST	CUSTNO	CUSTNO	T
ARCUST	PHONE	PHONE	F
ARCUST	ZIP	ZIP+CUSTNO	T
ARMAST	CUSTNO	CUSTNO+INVNO	T

Figure 1: **Dictionary entries** -- You can see how this approach imposes order on the design of the database by you and anyone else.

Sequence of usage

You should have a data dictionary manuscript for good system documentation. Programmers tasked with system maintenance or enhancement rely upon it. Creating a true data dictionary before you build databases facilitates the design process and automatically fulfills that documentation requirement. Creating the data dictionary after you build the databases only fulfills the documentation requirement, and programmer rarely enjoy creating documentation.

I've coded a simple system, DY-Dictionary, that facilitates the development of a database using the data dictionary methodology I've discussed. It's available for subscriber download at http://My.Advisor.com/doc/18900. The system lets you create the table and fields lists, then easily join them to design each table structure. Figure 1 shows a screenshot of the program.

The Table grid contains a list of system tables you define during the design phase. The Field grid contains the list of data elements along with their attributes. Adding and modifying the information is straightforward using the Add & Delete buttons and editing the value of each column. To begin, add a table such as VENDOR and enter a brief description.

You'll add many data fields during the initial phases of your project, but the type and width of each may change as the system evolves. To change the width of the CITY field from 20 to 30 bytes, scroll to that field and type in the new value.

To build the structure of a database table, merely highlight the desired table entry, the field to include, and click on Add New Column. Repeat this step for each field you add. I also defined a fourth table containing index tags with the additional capability to edit the tag name and index expression.

During the design process, you can print a Structure report (figures 2 and 3) for the database structure and associated index tags.

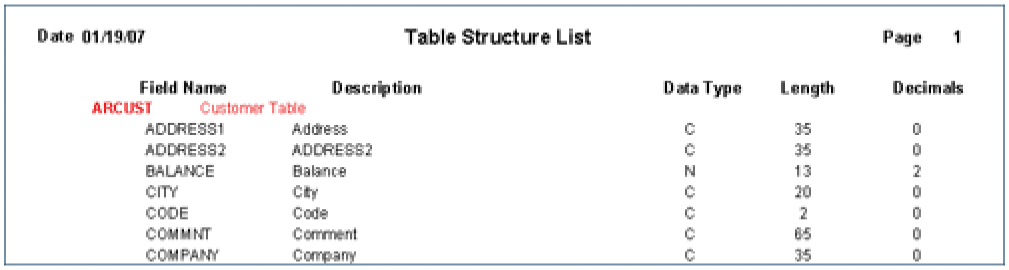
Design, create, document

My data dictionary methodology fulfills the design and documentation requirements of a project. But the DY-Dictionary utility I created goes one step further and actually creates the new database tables based upon the structure defined. Merely supply a valid subdirectory name, and the utility automatically builds the new tables for you. Further, if you change the structure, the target tables adjust automatically without loss of data. Database design, creation, and documentation are all rolled into one easy-to-use tool.

The DY-Dictionary utility creates and maintains FoxPro tables and is available free of charge at http://www.SouthBySouthwest.com. An expanded professional version of this utility (not free) manages Microsoft SQL databases and creates database containers. This Pro version lets your application operate with VFP or SQL databases using identical structures both maintained by a single data dictionary. The Pro version can also maintain separate data dictionaries for each of your separate application projects.

The payoff

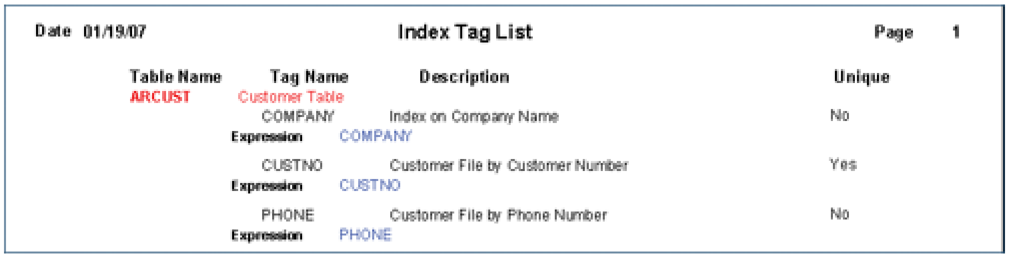
Using my data dictionary methodology takes the pain out of documentation because it's an automatic derivative of the design process. No longer do you have to transcribe the table structures into a text file and add supplemental descriptions for the documentation binder. Further, your databases will contain consistent data field names and attributes, which will make subsequent system maintenance much easier.
You can use a support utility such as DY-Dictionary instead of the FoxPro MODIFY STRUCTURE facility to create and maintain system tables. In a large development environment, programmers can now delegate the task of maintaining the dictionary definitions to clerical personnel.



Date 01/19/07 Table Structure List Page 1

Field Name	Description	Data Type	Length	Decimals
ARCUST	Customer Table			
ADDRESS1	Address	C	35	0
ADDRESS2	ADDRESS2	C	35	0
BALANCE	Balance	N	13	2
CITY	City	C	20	0
CODE	Code	C	2	0
COMMNT	Comment	C	65	0
COMPANY	Company	C	35	0

Figure 2: Table Structure report -- Itemizes each system table and the data elements contained therein.



Date 01/19/07 Index Tag List Page 1

Table Name	Tag Name	Description	Unique
ARCUST	Customer Table		
	COMPANY	Index on Company Name	No
	Expression	COMPANY	
	CUSTNO	Customer File by Customer Number	Yes
	Expression	CUSTNO	
	PHONE	Customer File by Phone Number	No
	Expression	PHONE	

Figure 3: Index Tag Report -- Itemizes each system table with the specified index tags.

ADVISOR GUIDE TO MICROSOFT VISUAL FOXPRO

September, 2007