



Breaking the Rules to Improve Performance, Part 2—Using 4D's Built-In Relational Search Engine

By: Walt Nelson

So that you will not have to refer to Part 1 to understand this discussion, we are repeating the Introduction that appeared in Part 1.

Introduction

In the database development community in general, and in the 4D development community in particular, we have certain “rules” that we observe and live by. Most of those rules are considered “good practice” in designing relational databases, and they stemmed from the ongoing research by experts in the field, such as E.F. Codd and C.J. Date. Some database designers regard those rules as unbreakable Commandments.

There are times, however, when you need to open your mind to possible approaches that achieve the desired results by *breaking* the relational rules. One such time is when you are having performance problems with a database—especially one that is used in a multi-platform environment.

Now that 4D Server supports multiple operating systems and multiple hardware platforms, performance will become more and more of an issue. If you write a database that is used by several different types of machines over a Wide Area Network (WAN), you will have to be much more concerned about performance than you were when the database was used in one building, on one platform, over direct an Ethernet connection. Performance that was “acceptable” or even “outstanding” over a local area network, can become “unacceptable” or even “unbearable” over a wide area network or through a dial-up modem connection.

If you have experienced performance problems with an application—and if you are willing to open your mind to “unconventional” solutions, then this technical note should benefit you.



Three Rules That We Will Break to Increase Performance

There are many rules that we 4D developers have learned to live by over the years. There are three rules in particular, that can adversely affect the performance of an application in a WAN or a dial-up situation.

- Don't use subfiles
- Don't use 4D's built-in relational lookups; write your own.
- Don't duplicate data—normalize

In this technical note, we will talk about performance-enhancing ways to break rule #2 (4D's built-in relational lookups) and rule #3 (de-normalized data). In a companion technical note this month, we discussed performance-enhancing ways to break rule #1 (subfiles) and rule #3 (de-normalized data). When we discuss each rule, we will first examine why the rule exists and then we will understand why breaking the rule, in certain cases, can give you geometric improvements in performance.

Rule #2: Don't Use 4D's Built-in Relational Lookups

As developers get more proficient in 4D development, they stop using 4D's built-in relational lookups, even though these lookups are significantly faster than any custom routines that a developer can create. Usually the reason that the developer started writing custom routines was to overcome the perceived "disadvantages and limitations" of 4D's built-in lookups.

Rule #3: Don't duplicate data; normalize.

To *normalize* data is to design the physical structure of the database so that, with the exception of primary and secondary keys, no piece of data is located in more than one place. For example, if you have a [Companies] file and a [Contacts] file, normalization rules dictate that you only store the Company Name in one place: the [Companies] file. In theory and in practice, there are very good reasons for normalizing data. Normalization simplifies data maintenance and reduces storage requirements.

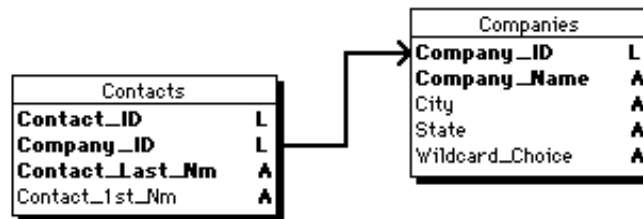
Disadvantages of 4D's Built-in Relational Lookup

Here are the disadvantages of 4D's built-in relational lookups from the developer's point of view:

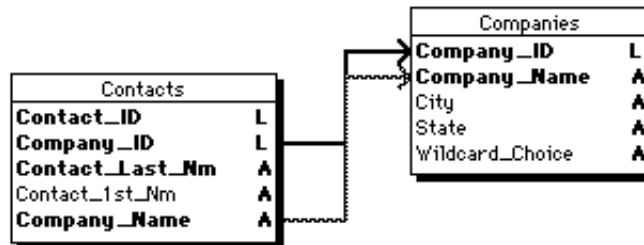
Names Do Not Make Good Primary Keys

Relational lookups are usually from a Secondary key in the Many file to the Primary key in the One file. A Primary key must be unique. Therefore, because of the chances for duplication, names usually do not make very good primary keys. Many developers

use a 'synthetic key', such as Customer_ID, Invoice_ID or Part_ID, as the primary key. Here is an example of a typical relation between files in 4D:



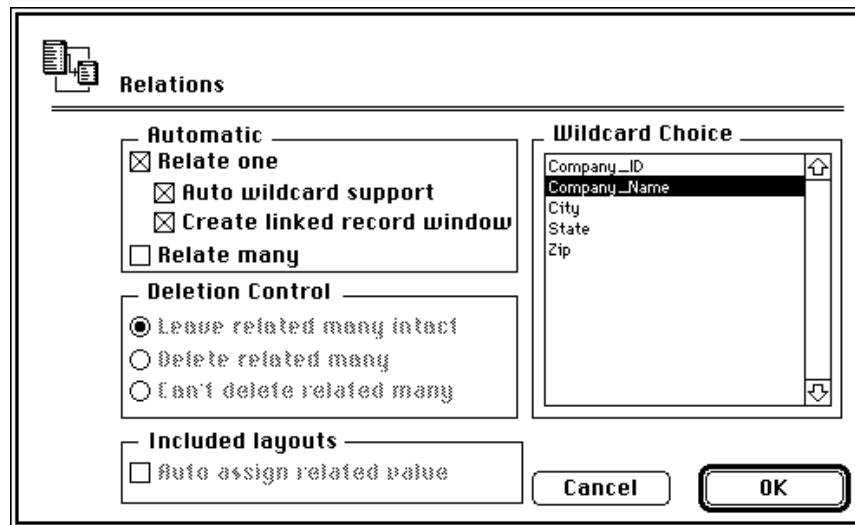
The field [Companies]Company_ID is the primary key of the [Companies] file; however, the users may not know the Company_ID. Rather, the users may want to search for the company by its name. In order to use 4D's built-in relational search, you would have to duplicate the Company_Name field in the Contacts file and then create a second relation between the two files. If you did that, your structure would look like this:



Although 4D allows you to create this kind of double relation between files, it is *not* a good idea to do so. You will get very unpredictable results when you perform relational searches and joins. Therefore, whenever the developer uses a synthetic ID as the primary key, rather than a name, the developer will probably conclude that 4D's built-in lookup is not appropriate; a custom relational lookup is a better choice.

Lookup Selection List Limited to Two Fields

Another reason that a developer may choose to create a custom relational lookup instead of using 4D's built-in mechanism is that 4D's lookup selection list is limited to two fields: the unique key field and a Wildcard Choice field. The following dialog lets the developer designate which field will be the Wildcard Choice column in the relational lookup selection list.



Sometimes, two fields do not provide sufficient information for the user to be sure which is the correct record. In the case of a Company, two columns would probably be sufficient. In other cases, however (e.g., Contact_Last_Nm and Contact_1st_Nm), two columns may not be enough. The user needs more information in order to decide which record is the correct one. In such cases, the developer will probably conclude that a custom relational lookup is the appropriate choice.

List of Choices is Not Sorted

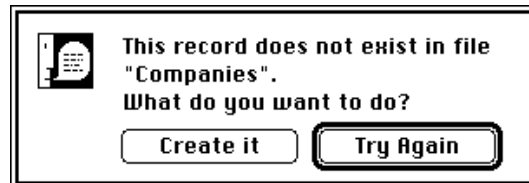
Another reason that a developer might create a custom relational lookup instead of using 4D's built-in mechanism is that 4D's lookup choice list is not sorted. 4D displays the list in the order that the records are physically stored in the database; the user must scroll through a random list and find the correct record.

No Control over the Look of the window

4D automatically determines the size, font and location of the built-in relational lookup window. Developers like to exercise more control over the interface than they have with the built-in relational lookup.

Loss of Control over New Primary Key Values

If the developer clicked the *Create linked record window* check box and the user enters a value that does not exist in the One file, 4D automatically assigns that value as the primary key and does not let the user change it.



If the user chooses to create a new record, the Companies data entry screen appears.

Company_ID	<input type="text" value="1029"/>
Company_Name	<input type="text" value="amer"/>
City	<input type="text"/>
State	<input type="text"/>

In the above example, the user entered "amer" as the search value, looking for American Airlines. The record did not exist, so the user chose to create it. 4D defaulted the primary key value to "amer" and would not allow the user to change the value. This behavior is not appropriate. In this case, the developer would probably conclude that a custom relational lookup is a better choice.

Advantages of 4D's Built-in Relational Lookup

Despite the previously mentioned disadvantages, 4D's built-in relational lookup mechanism has some powerful advantages that make it the right choice in certain situations.

Speed

The speed of 4D's built-in relational lookup is amazing. It is at least twice as fast as any custom lookup routine that the developer could write. In some cases, the performance difference is *several hundred percent!*

Simplicity

The developer can use 4D's built-in relational lookup without writing a single line of code. Even when you want to write code to overcome some of the disadvantages, the amount of code is much less than what would be required if you wrote a completely custom relational lookup routine.

Maintainability

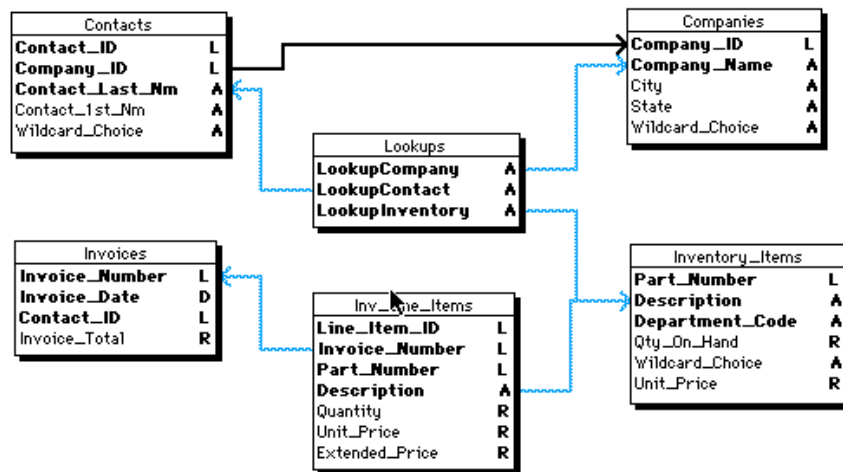
Generally speaking, the simpler the code, the easier it is to maintain and to understand. This can be a significant benefit in large projects, or in projects that are handed off from one developer to the next.

Speed

This factor can be so important in some situations that we want to emphasize it again. When the application is heavily used in an on-line real-time environment or when the user is connecting over a WAN or dial-up connection, speed can be a key factor in determining user satisfaction with the application. If users perceive the application as being "too slow," that perception can become a major issue, completely overshadowing the benefits of the application.

Implementing 4D's Built-in Relational Lookup

The speed factor alone makes it worthwhile for you to at least experiment with 4D's built-in relational lookup capability. The accompanying example uses 4D's built-in relations to implement many-to-one lookups in the following structure:



This is a simple Invoicing database that contains the following lookups:

- Company lookup from the [Contacts] input layout;
- Contact lookup from the [Invoices] input layout;
- Product lookup from the [Inv_Line_Items] included layout.

Setting Up the Relational Lookups

The example database shows you how to implement 4D's built-in relational lookups in a way that has the advantages of speed and simplicity, but at the same time overcomes some of the disadvantages that we previously mentioned. See the example database for more details.

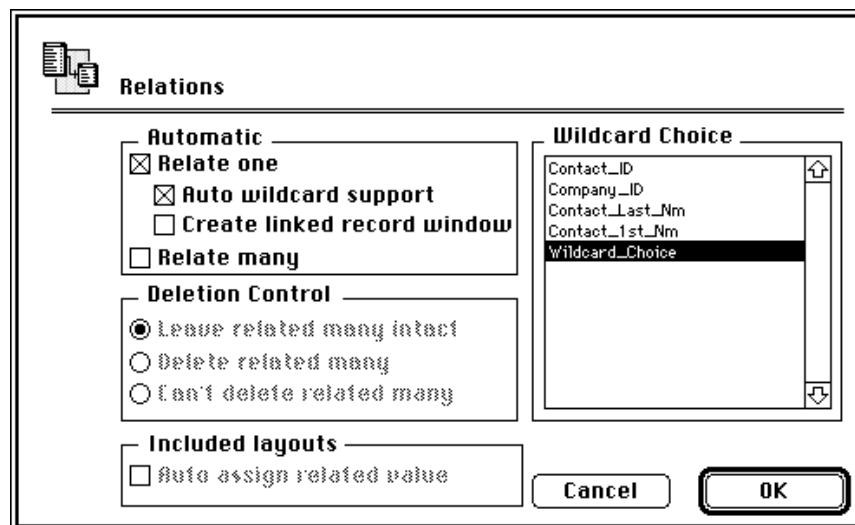
To implement this process:

Step #1: Create a [Lookups] file

This file will contain alpha fields that are used as "relational connectors" between files. This is how you avoid having double-relations between two files.

Step #2: Create Wildcard_Choice fields

In every 'One' file that you want to use for lookups, you create a field called [OneFile]Wildcard_Choice. You will concatenate several pieces of information into this field, and then use this field as the Wildcard Choice.



This is how you get around the limitation of only being able to show information from one additional field in the relational lookup. In the [Invoices]-to-[Contacts] lookup example, you will show two fields in the [Contacts] file, along with three fields from the [Companies] file. You will do so by concatenating information into the [Contacts]Wildcard_Choice field.



4th Dimension

Technical Note 96-32

Notice that you selected **Automatic Relate one** and **Auto wildcard support**. This means that 4D will automatically perform the lookup as soon as the user tabs out of the field in the related many file. Also, before it does the lookup, it will add the wildcard character (@).

Notice also that you did *not* choose the option **Create linked record window** in the above dialog. Because of the problem with partial search values (e.g. amer), you will have to write your own custom routine to create a related-one record if it does not already exist.

Step #3: Write Procedures to Concatenate information into the Wildcard_Choice field

Our example database pads the data with spaces to make each portion of the information a fixed length. In the Selection window, you can present the concatenated field so that it looks like several different columns. See the procedure that does this padding, *StringPadSpaces*, in the example database..

In the [Companies] file, we execute a procedure to parse information into the field [Companies]Wildcard_Choice. See the global procedure WildcardCh_Comp at in the example database.

Step #4: Place the Lookup Field on the appropriate Input Layout

For example, on the [Contacts] input layout, we place the field [Lookups]LookupCompany.

Contact_ID	<input type="text" value="Contact_ID"/>
To test the Company Name lookup, type an "a", then press the Tab key.	
Company_Name	<input type="text" value="LookupCompany"/> Company_ID <input type="text" value="Company_ID"/>
Contact_Last_Nm	<input type="text" value="Contact_Last_Nm"/>
Contact_1st_Nm	<input type="text" value="Contact_1st_Nm"/>
Wildcard_Choice	<input type="text" value="Wildcard_Choice"/>

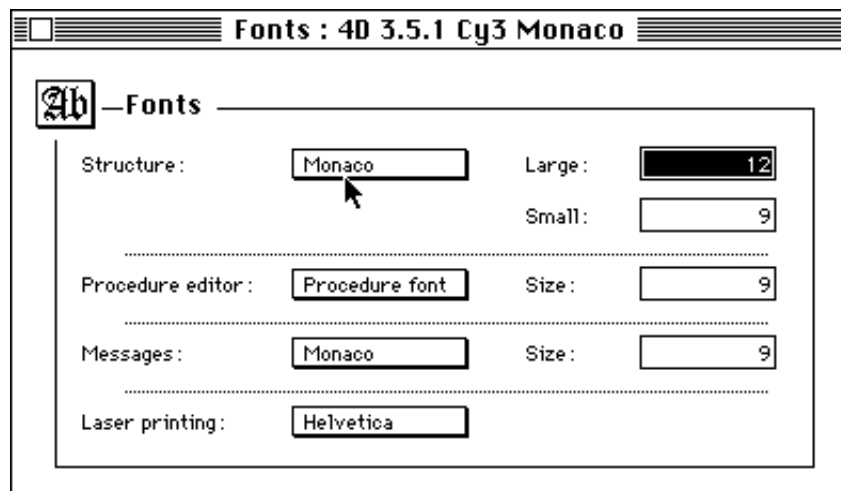
Step #5: In the Before phase of the input layout, set up data entry

See the layout procedure for the layout *[Contacts].Input f* in the example database.

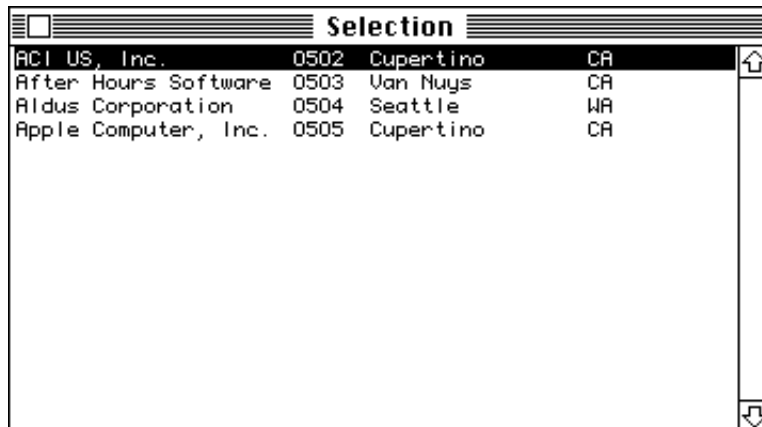
Note that you are creating a [Lookups] record. You will never save this record. You need to create it because technically, you are performing a Relate One between the [Lookups] file and the [Companies] file. This means that you must have a current record in the [Lookups] file.

Step #6: Use Customizer Plus to set the Structure Font to a mono-spaced font

When you concatenate information into the Wildcard_Choice field, you want the information to line up into neat columns. The Structure Font is the font setting that controls the look of 4D's relational lookup Selection window. Set this font to a monospaced font such as Monaco (Macintosh) or Courier (Mac & Windows).

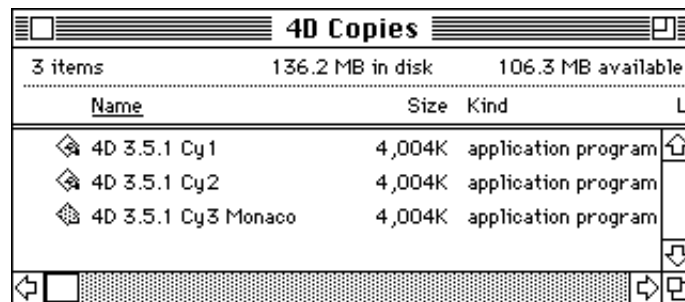


This setting will result in a text field that lines up perfectly, as you can see below.



There are only two fields in this selection window: Company Name and Wildcard Choice. The Wildcard Choice field is 30 characters: 6 characters allotted for the *Company_ID*, 20 characters for the *City*, and four characters for the *State*.

Note: The Structure font applies to several structure objects: Procedures, Structure window, and Selection window. Therefore, when you set the Structure Font to a monospaced font such as Monaco or Courier, you are also changing the font in the Structure window and the Procedure Editor. If you do not like to use that font for development, you can create a special copy of 4D for testing; this copy will have the monospaced font setting. Your other development copies of 4D will have the normal structure font setting, which is Geneva (Macintosh) or Ariel (Windows). Whenever you want to check the look of your relational lookup Selection windows, you can launch your application with the special (monospaced font) copy of 4D.



Name	Size	Kind	Location
4D 3.5.1 Cy1	4,004K	application program	
4D 3.5.1 Cy2	4,004K	application program	
4D 3.5.1 Cy3 Monaco	4,004K	application program	

When you deploy the application to users, you should Customize the compiled/merged application(single user) or Customize 4D Client (multi-user) so that the procedure font is a monospaced font.

Summary

By breaking the rules regarding relational lookups and de-normalized data, you can significantly improve the performance of your 4D applications.

Now that 4D Server supports multiple operating systems and multiple hardware platforms, running over WAN's and dial-up modem connections, *performance* will become more and more of a key issue. By using 4D's built-in relational lookups and duplicating certain data, you can improve the performance of your many-to-one relational lookups by as much as 100% or more.

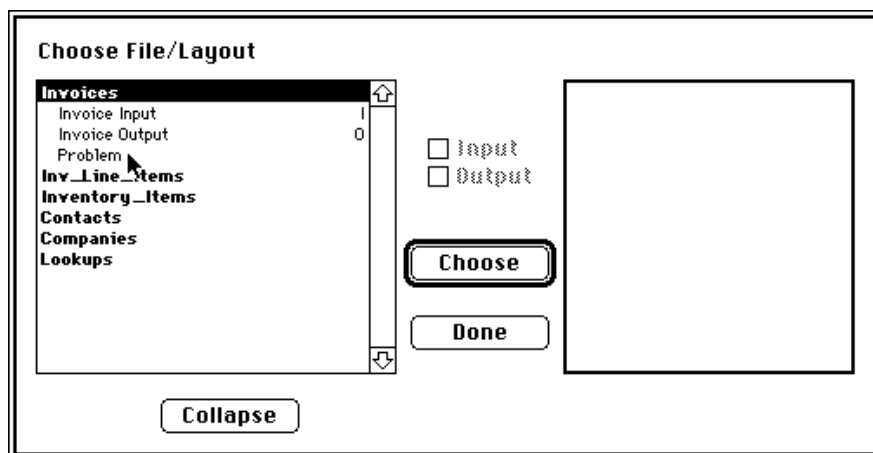
Appendix: About The Example Database

The example database *Performance Rules* contains examples of the performance-enhancing techniques covered in Technical Note 96-31 and 96-32.

The example database contains several layouts that use 4D's built-in relational lookups. When you launch the database, you are in User mode. The [Contacts] input layout contains a lookup into [Companies]. The [Invoices] input layout contains two lookups: one lookup from the [Invoices] file into the [Contacts] file, and another lookup from the [Inv_Line_Items] included layout into the [Inventory_Items] file.

When you look closely at the examples, you will notice that the link between [Inv_Line_Items] and [Inventory_Items] is direct; the [Lookups] file is not used as a relational link between the two files. There are two reasons for this:

- The Description field in the [Inventory_Items] file is unique, so it can be used for direct lookups.
- In the case of an included layout, the [Lookups] file concept will not work. You need to display multiple values for the Description (one for each line item), but you only have one [Lookups] record available for use. To see this limitation, go to the [Invoices] file, choose the layout [Invoices].Problem as the input layout, and enter multiple line items.



To understand how this overall technique works, study the following global procedures:

LookupCompany

StringPadSpaces



4th Dimension

Technical Note 96-32

WildcardCh_Cont

WildcardCh_Comp

WildcardCh_Invt