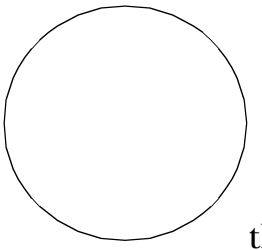
5

DEVELOPING RELATIONAL DATABASES



Flip back and take a look at Figure 1-1 again. Notice the area where all the ingredients for this hot chocolate recipe are listed (hot chocolate mix, boiling water and a mug)? Each of these ingredients is on its own line in a *portal*, a FileMaker interface element that allows you to see into another related database from the current database.

We weren't trying to trick you, but this recipe box database *is* a simple relational database system. There are two databases in this system, one called Recipes.fp5, the other called Ingredients.fp5. The Recipes.fp5 database has a *one-to-many relationship* with the Ingredients.fp5 database in that any one recipe can have many ingredients but any given ingredient only has one recipe.

Let's take a look at another example.

A Database of Your Friends

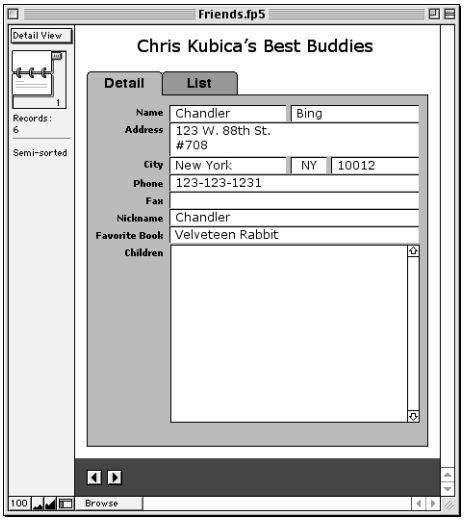
If you are keeping track of your friends on a personal database at home (so you never forget to send them birthday gifts ever again), such as the database shown in Figure 1-5 in Chapter 1, you might need only one database for all of the information about each friend. You would probably need fields for name, address, phone number, e-mail address, birth date, and so on.

However, if you wanted to start tracking related lists of information for each of these people, like their hobbies, or their children (and *their* birthdays) or their favorite books, or any other related list of information associated with each person, you would probably set up a related database.

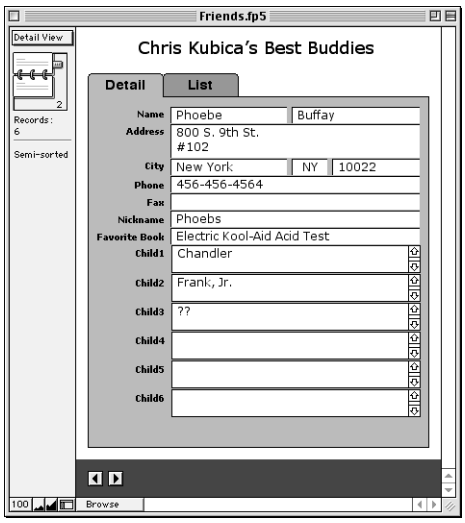
Alternatives to a Relational Database

Actually, you *could* do one of several things instead of using a relational model. Let's examine the possible scenarios using the situation where you wanted to keep track of all of the birthdays of your friends' children:

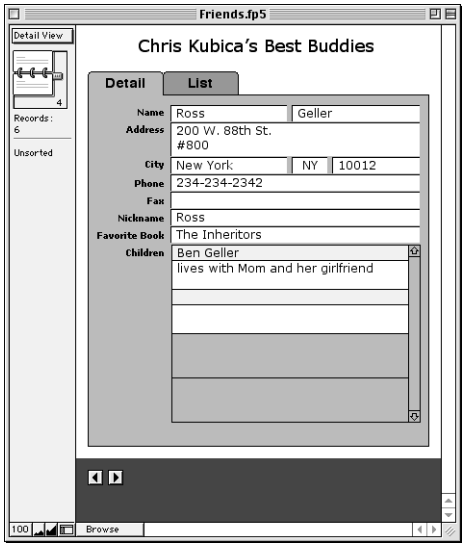
Scenario 1: Create a field in Friends.fp5 called Children (as shown below) and enter the names of everyone's child in this field, followed by carriage returns. You would probably make this field appear as a multi-line field on screen to accommodate all those carriage returns, and add a scroll bar on the Children field so that you could scroll through the contents of the field if it didn't entirely fit on screen.



Scenario 2: Create a field in Friends.fp5 for each child of every friend (shown below). You might guess that none of your friends would ever have more than six children, and create Child1 through Child6 fields. You might then stack these fields on screen and enter each child's name for each friend in the appropriate field.



Scenario 3: You might also create a related database called Children.fp5 with a Child and perhaps a Notes field in it, then add a portal to this database on the screen in the Friends.fp5 database (as shown below). For every child of each friend you would add a line in this portal which, in turn, would create a related child record in the Children.fp5 database which would always show up in that friend's Children.fp5 portal.



So what are the advantages of using the related database method versus the *flat-database method*, the method described in scenarios one and two, where all of the data in your database system is entered into only one database? First let's learn about the disadvantages of the flat-database method.

Disadvantages of Scenario One

In Scenario One, you have to lump all kinds of different data into one field, including a child's first name, last name, middle name, address, phone number, and so on. This is bad design which makes it almost impossible to automatically sort the children alphabetically in a list.

It is impossible to do a lot of things, actually, with this sort of scenario. For example, to alphabetize these fields you would have to go into this field and manually cut and paste each name in order. Too, this design doesn't force you to enter your data in any standard way by requiring data like first and last name be put into specific, separate fields. In fact, the Children field is completely free-form: You could enter last name/first name one time and first name/last name another. The field doesn't care.

Too, FileMaker limits text fields to only 64,000 characters so if you were tracking something more verbose than children's names (like a log of every time you telephoned someone) you would be in trouble after a while because you'd run out of entry space.

Disadvantages of Scenario Two

Scenario Two is a little different in that the data for each child *is* separated out a bit more in that each child is entered into a separate field. You could even add a Child1LastName field and split out the children's first and last names into separate fields to standardize data entry. But, again, you could never automatically alphabetize the names in a list or report, and a user could technically fill in the Child3 field and leave the Child1 and Child2 fields blank. And, for every friend that has only one child, you must always see five other blank fields on the screen that will never be filled in. This is a waste of valuable screen real estate.

But by far the biggest problem with scenario two would be if you acquire a friend with seven children. What would happen then? This scenario would leave you no place to enter information on this seventh child and would force you to add a Child7 field to the Friends.fp5 database.

While this might not seem like a big deal with this simple single-user system, consider an online store like Amazon.com. In a more complex system like this, any given customer's order might have one item on it, or 200 items! And what if for each item on the order you wanted to include four bits of data about the item, tracked as it is added to the order (for example, ItemID, ItemName, ItemDescription, and Price) so that the order can be properly invoiced and packed at the Amazon.com warehouse?

If someone were to order 200 items at Amazon.com and Amazon.com were to use scenario two, Amazon.com's Orders.fp5 database would need 800 fields just to track all the information on the ordered items for each order. And then what if someone were to order 201 items?! Start passing out the Excedrin Migraine.

The Solution: Scenario Three

Enter the relational database model. With this model, you simply set up a relationship from the Friends.fp5 database to the Children.fp5 database and put a portal to this database on a layout in FileMaker. Now, whenever you want to enter a child for

one of your friends, you just scroll to the next line in the portal and type in the child's information. No matter how many children any friend has, whether they have one or twelve, there's always another portal row to enter the name into.

In the relational model, Amazon.com can have a related database for all the items on a given order. When someone clicks the order button on an item, the database simply "goes to last portal row" and enters the item on the order. It doesn't matter how many times the user clicks. The related database never runs out of room in a field as it would in Scenario One, and it never runs out of fields as it would in Scenario Two.

Another good reason to use the relational model is because it eliminates the need for duplicate or redundant data. On a flat database Amazon.com invoice, every time a user wants to order something they'd have to re-enter all their contact information first, which would be a big turn-off. But if a customer has a unique customer ID and creates an order, all of that customer info can be "looked up" automatically, via a relationship, and placed on the order, forever preserving the contact info just as it was when they submitted the order.

The Relationship Between Databases

Let's explore the mechanics behind a relationship between two databases. An *E-R* or *Entity-Relationship Diagram* is a diagram that depicts all of the databases in your database system and their relationship to each other. Figure 5-1 shows an E-R diagram for our "Friends and Family" database system.The one-to-many relationship between Friends.fp5 and Children.fp5.

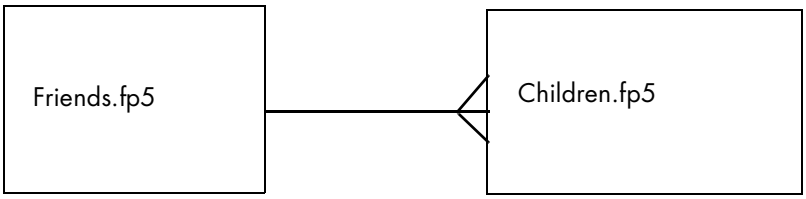


Figure 5-1: The one-to-many relationship between Friends.fp5 and Children.fp5

Friends.fp5 is represented here as a rectangle, as is Children.fp5. The forked line that connects the two represents the one-to-many relationship from Friends.fp5 to Children.fp5. Any given friend can have many children, but any child only has one parent in Friends.fp5. (This assumes that you don't list both parents of a child as separate records in Friends.fp5; a situation that would call for a many-to-many relationship).

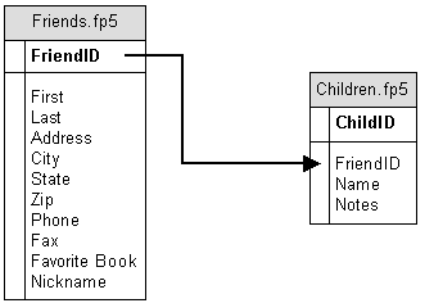


Figure 5-2: The one-to-many relationship between Friends.fp5 and Children.fp5, showing the fields that this relationship is based on.

Figure 5-2 is another way of looking at the relationships between these two databases. In it you can see the fields that this one-to-many relationship is based on, which is the FriendID field. (You were introduced to the concept of an ID or serial number field earlier.) The FriendID field exists in Friends.fp5 to uniquely identify each friend in the database. In fact, each time you create a new record in Friends.fp5, the FileMaker database automatically enters the next sequential serial number into this field for the new record.

Primary Keys

The FriendID field is known as the *primary key* of Friends.fp5, which means that this is the field used to uniquely identify each record in a given database. In this case, it is the main field used to relate a friend record in Friends.fp5 to a set of records in the Children.fp5 database. (You could use the combination of someone's first and last name as the primary key in a database, but if you did this, luck would have it that eventually you would acquire *two* friends named Rupert Pupkin and then this primary key would no longer be a unique identifier.)

Note that since FriendID is the unique identifier for a record, it is a *big* problem if somehow two or more of your Friends.fp5 records end up with identical FriendIDs. If this were the case, the records with the same FriendID could not be uniquely identified. Thus it is usually best to allow your database to automatically generate unique IDs. They never even have to show up on the screen (the numbers are usually meaningless to a user, anyway) but could simply exist in the background to maintain the integrity of your database relationships.

Foreign Keys

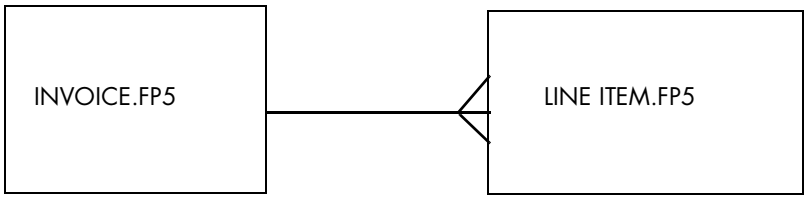
FriendID also exists in the Children.fp5 database. In this database, FriendID is called the *foreign key* because it matches the primary key in another database (Friends.fp5). Children.fp5 also has its own auto-entered primary key, ChildID, which you can use to establish relationships from Children.fp5 to another database that you might add to the database system in the future.

Basically, when you enter data into the portal to Children.fp5 on the entry screen for Friends.fp5 (the screen shown earlier for Scenario three) here's what happens: As soon as you start typing a child's name into a blank line in the portal, FileMaker creates a new record in Children.fp5, auto-enters the next sequential ChildID value (to make the new Children.fp5 record unique) and also *automatically* enters the FriendID of the friend record you are currently viewing into the FriendID field of the new child record you are creating in the portal.

In FileMaker, whenever you are entering related data into a portal like this, the foreign key of the related database is always entered for you automatically. It makes sense: in order for the record you are creating in the portal to maintain its relationship to the current friend record, the current friend record and the new related child record must have the same FriendID.

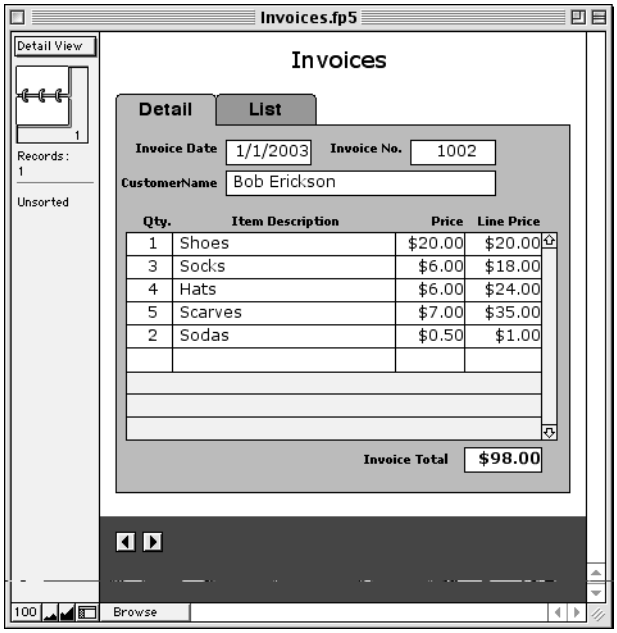
Invoice Database Systems

Another common example of a simple one-to-many relationship is that of an invoice database system. An invoice database system is usually composed, at its core, of two databases, Invoices.fp5 and LineItems.fp5. One invoice can contain an infinite number of items being ordered, but any given line item is only associated with one invoice (assuming that partial payments are not allowed). The relationship would look like this:



The invoice database would have fields like InvoiceID, InvoiceDate, customer contact information, and the customer's method of payment. Each line item on an invoice would contain information on the products being ordered: a unique LineItemID, a ProductID, Price, Description, and Quantity, and probably a calculated cLinePrice field that would multiply each line's quantity by its price to get the total price for that line.

Below is what a simple invoice database might look like in FileMaker Pro. Note the line items portal, where you can see into LineItems.fp5. A user can enter items for this order into the portal and each line item will receive the InvoiceID of the current invoice on the screen when entered.



Relational databases can be very complex, but once you know how to set them up in FileMaker, it's not too bad. In this book, you will learn several other types of relationships as well, including one-to-one, many-to-many and self-join relationships. Relationships add real power to a database system, the benefits of which you will soon discover.

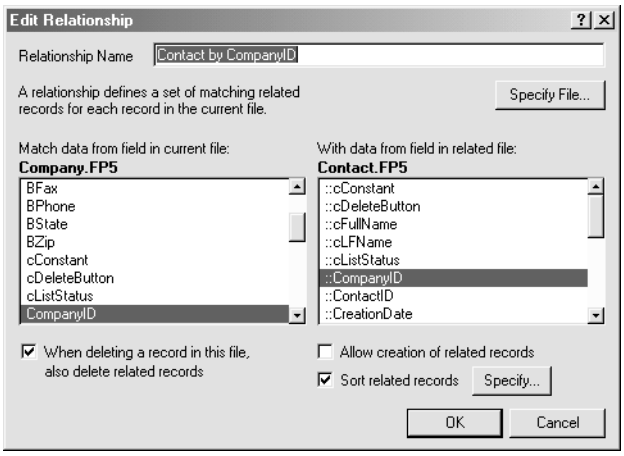
Relationship Set Up

Now that you understand the basics of the relational database model, let's look at how you set up a relationship in FileMaker Pro. To begin, open the Computer Shop folder in the Chapter 5 folder on the CD-ROM then double click on the Company.fp5 database. Navigate to the Contacts tab and switch into Layout mode.

Note that there's a portal into a related database here already, to Contact.fp5. In order to see the definition for this relationship select Define Relationships from the Database menu; the Define Relationships dialog appears.

You'll see in this dialog that the relationship to Contact.fp5 is already defined, but take a moment to explore this dialog box before you go any further. Note that you can sort relationships by clicking on the columns or by changing what's in the View By drop down menu. You can create new relationships with the New button, edit the highlighted relationship with the Edit button (or just double-click the relationship), duplicate the highlighted relationship with the Duplicate button and delete the highlighted relationship with the Delete button (remember, this is permanent). The Done button closes the dialog. The information shown for each relationship is the name you specify, the fields involved in the relationship and the database being related to from the current database.

Go ahead and double-click the relationship called “Contact by CompanyID”. The following Edit Relationship dialog will come up:



The name of the relationship is in the box at the top (see Chapter 14 for naming conventions here). The Specify File button allows you to navigate to the database you want to relate to. In this case, Contact.fp5 is already selected and its fields appear in the lower right window. Company.fp5’s fields are in the lower left window.

NOTE *In order to define a relationship to any database, you must know its master password. If the related database doesn’t have the same password as the current one, FileMaker will ask you to enter its password.*

To pick the fields of the relationship that you want to use, scroll through them and single click a field to highlight it. Here, CompanyID is already highlighted on both sides of the relationship as these contacts are “children” of their “parent” company record.

NOTE *On the right side of a relationship, the field you choose must be stored or indexed. Thus, the right side can’t be an unstored calculation, a global field, a summary field or a container field.*

Here’s what the checkboxes at the bottom of this dialog do.

When deleting a record in this file, also delete related records

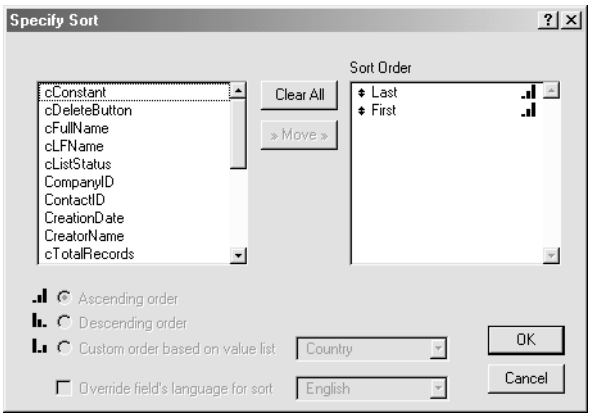
If this box is checked, deleting the master record will delete all related records. For instance, if you delete a company record, all related contact records would be deleted too. You may want this to happen sometimes (for example, when deleting an invoice you can probably delete its related line items) but be careful when checking this box: The related records are deleted without warning.

Allow creation of related records

If this is checked, you can put your cursor into an enterable field in the portal and type in a related record. When unchecked, a portal to a related database is read only.

Sort related records

Checking this box and clicking the Specify button brings up the following Specify Sort dialog:



Use this dialog to specify how related records will be sorted in a portal. For example, for the relationship to Contact.fp5 that we looked at above, the related records will be sorted by last name first, then by first name, in ascending order.

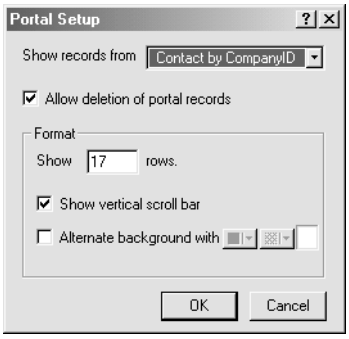
To specify the sort for a relationship, single click a field in the left portal, then select "Ascending order" (A to Z), "Descending, order" (Z to A), or "Custom order based on value list" (which sorts related records in the same order as the selected, pre-defined value list), then click the Move button. The field will move to the Sort Order box. You can also specify how each field sorts by selecting a field in the Sort Order box and changing its method of sorting with one of the radio buttons.

When checked, the "Override field's language for sort" option says to override the language of the field's index (see Chapter 2).

Normally, you'll keep the language set to your language (English, for example) but you may want to change the language to ASCII which takes characters like punctuation marks and other high ASCII characters into account in the sort.

Portal Set Up Basics

Now let's set up the way a portal to related records functions. Go back into Layout mode on the Contacts tab of Company.fp5. Double click on the grid-like Contact by CompanyID portal and you'll see the Portal Setup dialog:



Portal Setup Dialog

Here you can specify which relationship to use in the "Show records from" drop down menu. If the "Allow deletion of portal records" is checked, users will be able to highlight a portal row and delete the related record.

Entering a value in "Show *n* rows" specifies how many related records will appear in the portal on the layout (but not how many related records there can be, which is virtually infinite) as well as how much vertical space the portal will take up on a layout. Checking the "Show vertical scroll bar" box will add a scroll bar at the right of the portal so you can scroll up and down through all related records, regardless of whether they surpass the number of rows in the "Show *n* rows" field.

If you check "Alternate background with" and then select a color and pattern, you set the appearance of every other row in a portal. This is handy in large portals because it makes each portal row easier to read. Remember though that any color specified here will print on a print layout, and that the color and pattern selected here are the alternate color of any color or pattern specified on the portal object itself in Layout mode.

Cancel out of this dialog, and note that in order to show related data in a portal, you've got to arrange the related fields over the top of the portal on the layout, *not* under it. Use the field tool to drag a new field to your desired position on the layout and specify the field using the same relationship as the portal (or you'll get very strange results). Here, you've already got the cFullName, Title, Phone and Email fields.

You only need to put related fields on the first visible portal row. In the other modes, whatever you put in the first row will replicate automatically on all other visible rows. You can put buttons, lines, or other objects on the first portal row, too, and they'll also be replicated on every portal row. Vertical lines can be drawn across the whole portal, crossing all rows, to add a nice divider between fields in the portal.

NOTE *To number records in a portal, place "@@" in the first portal row by going to Record Number Symbol from the Insert menu.*

When You Don't Need a Portal

Go into browse mode and click on the first contact of record #2 in this database, which should be Chris Kubica (though any contact from any company will do). A script runs and takes you to the contact you clicked in the Contact.fp5 database. You're now looking at the detailed information for that contact.

Now go into Layout mode on the General Info tab of Contact.fp5. Note that there are several local fields at the top (first name, last name, and so on) and several related fields in the middle (marked by the :: at the start, such as ::bCompanyName) that are *not* in a portal. Why are these related fields not in a portal? Well, related fields don't have to be in a portal and usually are not when you're showing related data in a many to one relationship. There is only one company for this contact, whereas there were many contacts for this contact's company (thus the portal back on the Contacts tab of Company.fp5).

Here, all these fields show the related company information for this contact. This information does not need to be stored here in the contacts database because it would be redundant. This makes sense, because if you change the information for a company, like its main phone number, you would like that information to automatically update for all of its contacts, rather than have to update each contact manually. Now, of course, if contacts in your world usually have their own phone numbers other than the main company line, you would also give them a LocalPhoneNumber field in Contacts.fp5 (as you have here).

All of the grayish, non-editable company-related fields here are for a user's reference only. You could allow editing, but if a user edited the local company phone number from *here*, they could be unwittingly editing the master company phone number for all the company's contacts by mistake. So be careful when granting access to non-portal related fields on a layout.

Lookups vs. Related Data

In Chapter 2, you learned about the "lookup" auto-entry option for a field. Now it's time to learn when to use lookups when building a relational database system.

To recap, a *looked-up valu*e is a bit of data that is copied from a field in another database via a relationship. If you've got an invoice database with an associated line items database, whenever you add a product as a line item on an invoice (the customer is buying it), you want certain fields to automatically "look up" certain bits of data from a products database, like the product's name, price, or description, so that you don't have to look them up yourself. Perhaps more importantly, you want your historical information to be preserved on the looked up fields, regardless of any changes made from the "master" data source fields that the lookup fields look up from.

In the case of our related company information, these are dynamic, related fields that update automatically if company information is modified, and these changes are okay, even desirable. But in the case of something like an invoicing system, auto-updated product pricing on an invoice would not be a good thing.

For example, click on the Invoices tab in Contacts.fp5 and flip to a record that has a related invoice in the portal. Click on an invoice that has a total Order $ greater than zero. You'll find yourself in the Invoice.fp5 database on the General Info tab. Here you'll see all of the same company and contact fields that you see on the contact's record, but the fields here are local lookups, not related fields.

Now go into Layout mode and double click on one. Yup, local fields. Why? Well, if ever the contact or company information were deleted from the database, you wouldn't want data in dynamic, related fields on an invoice layout to just vanish into thin air; you want to preserve it forever as a history of who ordered what and when.

So when a new order is created and a contact selected, all the address/phone information is looked up via a relationship to Contact.fp5 and pasted, really, into the same-named local fields in Invoice.fp5. Now, even if the contact changes his address (and you've updated her record in Contact.fp5), and orders again, you'll have one invoice record with the old address and one with the new. This way, you'll always have an accurate history of what was sent where.

Now, if you go to the Line Items tab in Invoice.fp5 and go into browse mode, you'll see another important use of lookup fields. Here's where you select what is being ordered by choosing products in a portal.

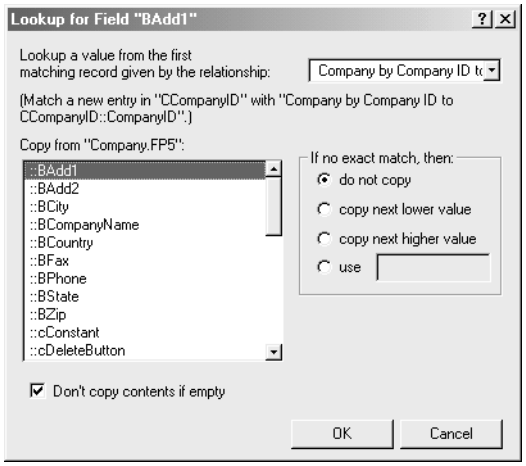
Go ahead, add a few to this portal in the next available open portal rows. As you are selecting products in the portal, all the fields in InvoiceLineItem.fp5 (like price and other descriptive information), the related database you're working with here, are auto-populated via auto-entered lookups. After an order's complete, you would print a copy and send one to the customer for payment, then enter that payment information on the payment tab. If the prices or product information ever changed in Product.fp5, it would be okay because the lookups in InvoiceLineItem.fp5 would preserve the old, looked up information before the product was updated.

However, if the fields in InvoiceLineItem.fp5 like price (called List in this database) were dynamically linked to Product.fp5's data (by being defined as an unstored calc like Products by ProductID::List), updating the list price in Product.fp5 would also automatically update the list price on all line items for all invoices in the system, changing the balance due, the invoice total, and other fields. Basically, you would have a big reconciliation mess on your hands.

Setting Up Lookups

Go to the General Info tab in the Invoice.fp5 database again and go into Layout mode. The BAdd1 (billing street address line 1) field is a lookup to Company.fp5.

Now access the Define Fields dialog and double click on this field. Go to the Auto-Enter tab and click the Specify button next to the Looked-up value checkbox and text. You'll see the Lookup dialog for BAdd1, which looks like this:



To define a lookup, all you do is specify which relationship you're going to use (via the drop down at the top), pick the field whose value you will be copying to the current database field (in this case BAdd1), and then specify what happens if there isn't an exact match via the relationship you've selected.

In this case, there will always be a valid company associated with the CompanyID of any invoice record, so if there isn't an exact match it's not that important. But in other cases where, say, the records on the side of the relationship where the data is being copied from change a lot (get deleted, have their primary key altered), you must specify what happens when there's no exact match, using the radio buttons in the middle right of the dialog. Here's what the different options mean:

- *do not copy:* Does not copy anything into the field, leaving its contents as is.
- *copy next lower value:* Copies the next alphanumerically lower value from the related database. "No zip code 47906? OK, give us 47905."
- *copy next higher value:* Copies the next alphanumerically higher value from the related database. "No zip code 47906? OK, give us 47907."
- *use __*: If there's no match via the relationship, it will use whatever you type into the box. "No product description for the Tiger Lilly Bulbs product? OK, paste in 'N/A'."

The checkbox "Don't copy contents if empty" says that if there is a match, but the matched field has nothing in it, the "nothing" won't be copied over into this database field. This seems irrelevant, but if you ever do a relookup (see Chapter 4), you may have a value in the field now that you don't want overwritten with nothing.

NOTE *If there's more than one matching related record via the relationship used in the lookup definition, FileMaker will paste in the value from the record that would appear first in a portal to that relationship. If there is no sort order for the relationship, this will be the first record created. If there* is *a sort order, this will be the first record in that order.*

From the Field: Advanced Portal Techniques

Now that you know the basics about relationships, here are some field-tested tips and tricks you can implement on your systems. Each technique has an explanation (some long, some short, depending on the technique at hand) and a set of example files on the CD-ROM for you to dissect and explore. You intermediate to advanced developers might find these useful.

Many to Many Relationship

On the CD-ROM: In the Many to Many folder of the Chapter 5 folder

Primary Database: Pets.fp5

The Problem: You can only imagine how many times the following dialog has been exchanged between FileMaker Pro developers and their users:

[Developer sits comfortably in an Aeron® chair in front of a Strawberry iMac, carefully monitoring a fictional client's FileMaker Pro Server's cache hits]

[Enter JOHN DOE USER, Developer's client contact and novice FileMaker Pro user]

JOHN: "Say, Developer, I've got this database called Pets.fp5 and another database called Supplies.fp5. Now I know how to create a portal in Pets.fp5 so that I can see all of the supplies linked to a pet. But how in the heck can I create a portal to link many different pets to many different supplies?"

DEVELOPER: What you need, my friend, is a many to many relationship.

JOHN: My many relationships are going just fine, thank you very much.

[Developer sighs a tired sigh]

Rising Action

So John's got a Pet data entry layout where he sees basic information about one of the pets he sells, an albino gerbil, and a portal where all the supplies that he needs to sell with the gerbil, the "gerbil set", if you will, are listed. Now, if he creates a new record in Pets.fp5 for the Syrian hamster he also sells and tries to enter in the portal the supplies associated with the hamster via a drop down list populated by the Supplies.fp5 database (many of which are similar to those in the gerbil set) everything seems to work fine from this point of view.

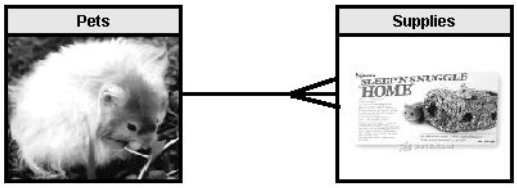
But the next morning, when John goes over to the Supplies.fp5 database because he needs to change the selling price and description of the carrot-shaped chew sticks supply record, there seem to be duplicate records all over the place! O! O! O!

What went wrong? How did John manage to muck up his system and add a bunch of duplicate supplies to his Supplies database? In a nutshell, John is trying to work as if he has the functionality of a many to many relationship when in reality he only has a one to many relationship. John wants to be able to link many of his supplies to many of his pets and vice versa.

But before we tackle that, let's define our terms.

The One to Many Relationship

Before John ever comes to Developer for help, he has a simple database structure that looks like this:



The forked line between Pets and Supplies means that there are many supplies linked to every one pet, but each supply has at *most* one link to a pet. This is a one to many relationship. Another example is how John's wallet has many colorful credit cards in it, but each credit card only lives in one wallet (John's).

The Solution

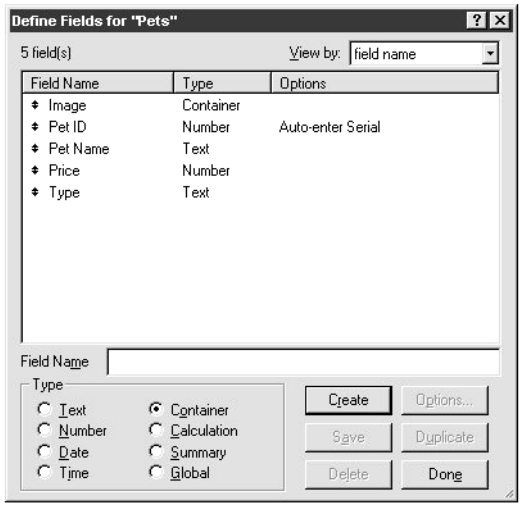
The solution is a Many to Many Relationship. What John actually want is a structure that looks like this:

Here, there is Pets.fp5 database on the left, a Supplies.fp5 database on the right, and a database called PetSupplies.fp5 in the middle. What's it doing in there? And how did it get in there? Basically, PetSupplies.fp5 is, as many Computer Science major database specialists will call it, a transaction database (or a junction database): a database that keeps track of all the links between Pets.fp5 and Supplies.fp5. No relationship between Pets.fp5 and Supplies.fp5 exists, directly, but the PetSupplies.fp5 database breaks down this "many to many relationship" into two manageable one to many relationships so that many pets can have many supplies. FileMaker developers call this a *join database*, and it is a truly magical thing. Are you still with us? (John is.)

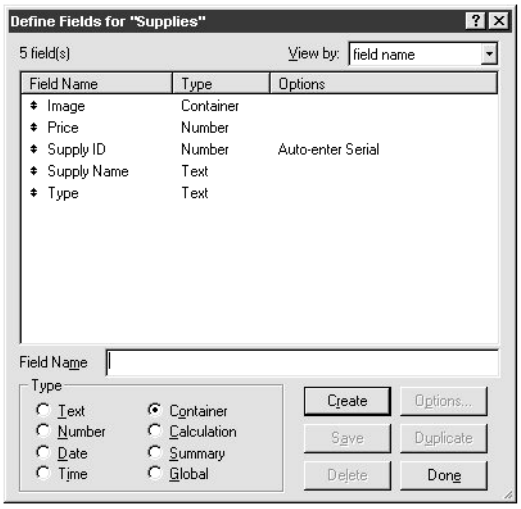
Setting Up a Join Database

Let's sit down with John and create this pet store system for him from scratch.

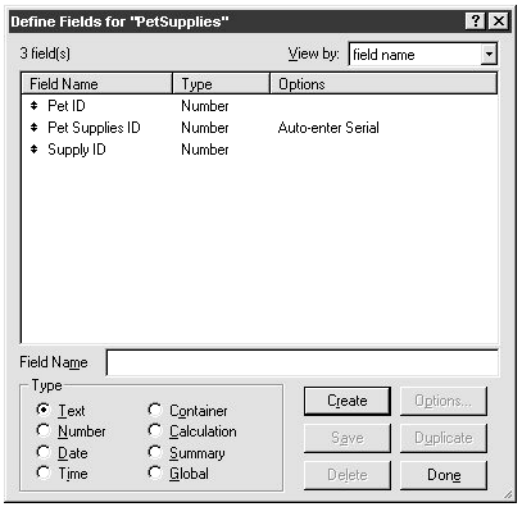
1. First, create a Pets.fp5 database with the following fields:



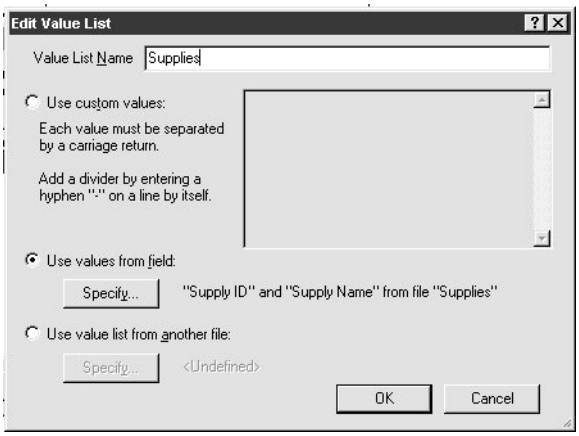
2. Next, create a Supplies.fp5 database with these fields:



3. Create a PetSupplies.fp5 database with the following few fields:



4. Now go back to your Pets.fp5 database and create a basic data entry screen, then go into Define Relationships and define a new relationship to the PetSupplies.fp5 database with Pet ID highlighted on both the left and right. (Make sure to check "Allow Creation of Related Records" and "When Deleting a Record in This Database, Also Delete Related Records.") Click OK.
5. Add a portal to the data entry layout using the relationship just created and put the Supply ID field into it, then define a value list in Pets.fp5 that looks like this:



and attach it to the Supply ID field in the portal to PetSupplies.fp5.

Now is a good time to add a few unstored fields to PetSupplies.fp5 that will make using this system more intuitive and much easier to use for John and his clerks.

1. First, create two relationships in PetSupplies.fp5, one called Pets by Pet ID (relating Pet ID to Pet ID) and another called Supplies by Supply ID (relating Supply ID to Supply ID).

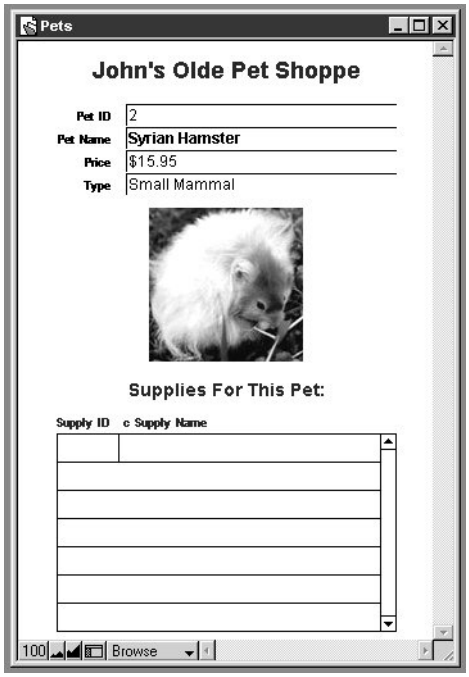
2. Next, create two calculated fields, as such:

   c Pet Name, an unstored text field, defined as: `Pets by Pet ID::Pet Name`

   c Supply Name, an unstored text field, defined as: `Supplies by Supply ID::Supply Name`

NOTE *Although you might like to make these fields stored because there's a good chance that users will want to find based on this field (i.e. which small mammals come with water bottles?), some calculation fields can't be stored and calculation fields that are based on fields from a related database (as these are)* must *be unstored. In fact, if you attempt to change them to stored, you will get a message that it isn't allowed.*

3. Go back to Pets.fp5 and add c Supply Name to the PetSupplies.fp5 portal, to the right of Supply ID. You should now have a simple layout that looks something like this:



The Climax

Now, if we've done everything correctly, you should be able to go into browse mode, click in the Pet ID field in the PetSupplies.fp5 portal, and select different supplies to associate with any pet! Half of the many-to-many relationship is complete ready to go. Now we need to set up a similar layout with similar functionality for Supplies.fp5.

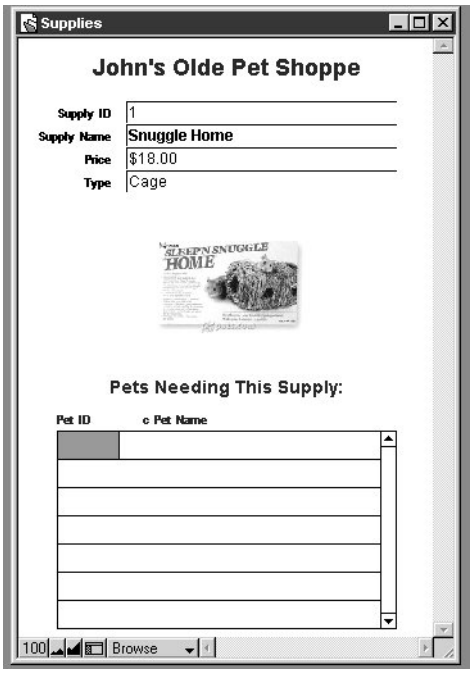
1. Go into Supplies.fp5 and create a similar basic data entry screen (use a different color background, too).
2. Next, go into Define Relationships and define a new relationship to the PetSup-

plies.fp5 database with Supply ID highlighted on both the left and right. (Make sure to check "Allow Creation of Related Records" and "When Deleting a Record in This Database, Also Delete Related Records.") Click OK.

3. Add a portal to the data entry layout using the relationship just created. Add the Pet ID field into it, and add the c Pet Name field you created earlier to the right of Pet ID.
4. Now define a value list in Supplies.fp5 that looks like the following and attach it to the Pet ID field in the portal to PetSupplies.fp5:

I05-17.tif TK

You should now have a simple layout in Supplies.fp5 that looks something like this.



Again, if it's set up correctly, you should be able to go into browse mode, click into the Supply ID field in the PetSupplies.fp5 portal and select different pets to associate with any supply you fancy!

Dénouement

Now that you've set up a basic many to many relationship, here's one more handy technique that will help you make John's or any client's life much easier and make this solution much more powerful and easy to use.

Navigation

Wouldn't it be nice if, say, you were sitting on the Albino Gerbil record and you wanted to quickly jump to the Chew Sticks records to edit it and then jump back to the Albino Gerbil record again? You can! Here's how.

1. Go into PetSupplies.fp5 and set up two scripts.
2. Hop to Supply Part 2, defined as:

   ```
   Go to Related Record ["Supplies by Supply ID"]
   ```

3. Hop to Pet Part 2, defined as:

   ```
   Go to Related Record ["Pets by Pet ID"]
   ```

4. Now go into Pets.fp5 and create a script called Hop to Supply Part 1, defined as:

   ```
   Go to Related Record ["PetSupplies by Pet ID"]
   Perform Script [Sub-scripts, External: "PetSupplies.fp5"]
   ```
   (Select the Hop to Supply Part 2 script)

5. Last, go into Supplies.fp5 and create a script called Hop to Pet Part 1, defined as:

   ```
   Go to Related Record ["PetSupplies by Supply ID"]
   Perform Script [Sub-scripts, External: "PetSupplies.fp5"]
   ```
   (*Select the Hop to Pet Part 2 script)*

Now all you have to do is go to the portal in both the Pet and Supply database and add an icon or button to the right of the portal (as shown at left) and link the appropriate script to it. Now you can click back and forth between pets and supplies with ease.

Epilogue (Summing Up)

John, the fictional client, has asked us to sum up by expounding upon the tremendous advantages a many to many relationship gives to his database. Well, we can tell you that with this type of relationship, the possibilities are almost endless.

First of all, it is sound database design, and won't leave you duplicating records in a way that will drive you nuts. And you'll be able to link many contacts to many projects, many products to many parts, or many students to many classes, many everythings to many everything elses. You can hop back and forth between all of these records with ease and in a way that is perfectly seamless to the user.

To add icing to the cake, you can do quite a bit of sophisticated reporting in PetSupplies.fp5 now that all the transactions that join the two original databases together are in one place. And you didn't think wonderful client relationships were possible, did you?

Stacking Portal Fields

On the CD-ROM: In the Stack Select Dupe folder the Chapter 5 folder.

Primary Database: Solution.fp5, layout ONE.

The Problem: You have limited space in a portal and you would like to have an ID field with a value list attached to it be hidden from the user's view (who doesn't care about the ID value), showing just the descriptive value of the related record. Another problem might be that you want a field in a portal to change to a certain color depending on the value in the field.

The Solution: Stack fields in the portal and play with their transparency. See the example database for more specific information. Information on setting this up is included in the solution database itself.

Duping A Record With All Its Related Portal Records Intact

On the CD-ROM: In the Stack Select Dupe folder the Chapter 5 folder.

Primary Database: Solution.fp5, layout THREE.

The Problem: Tons of in-house developers and intermediate users request this feature, because they can't figure out how to do in on their own. They want to duplicate an invoice or purchase order with a button that also duplicates all associated line items and (sometimes) contacts for that respective document, when there's more than one contact. Usually, users can't figure out how to do this because they don't have experience using global fields or constant relationships, or they're not too handy with the ScriptMaker even though you all know that the ScriptMaker can be a close, lifelong companion. This solution will put all of those questions to rest at last.

The Solution: Setting up this technique is trivial. Once you've set up a constant relationship, all you need to add is two scripts, one in your Line Items database, the other in the Invoice (here called Solution.fp5) database.

The script in the Line Items database (called, in the example databases, Support2.fp5) consists of several simple steps. First, the window is frozen, then Browse Mode is invoked. The Layout #1 layout is invoked, and you are taken to the first record in the found set, then the gRecordNumber field is set to the record number of the first record in the set.

Next, you enter a loop where, cycling through every record in the found set, you dupe the record, set the InvoiceID of the record to the ID of the newly created master InvoiceID just created in Solution.fp5, and then omit the record. Then you go to the next record in the found set, set the gRecordNumber field again and continue through all the line items until all are duped. Then you exit the loop.

The script in Solution.fp5 is just about as simple. First, you enter browse mode and isolate the current record that you want to duplicate into a found set of one. Next, you go to related records (show only related records) to isolate the line items

that you want to duplicate, then duplicate the record. After that, you set the g Invoice ID to the newly created Invoice ID so as to transfer the value to your newly imported line items.

Once you've performed the related script created above in the Line Item database, you'll probably want to add an Exit Record script step to make sure the user is left in the Invoice database, rather than Line Item. Or add in a Show All Records step like you did here. And you're finished.

NOTE *You could also do dupes of all the related contacts, purchase order line items, or any group of other related records hanging off of Solution.fp5 (or line items even, in a nested duplication process) using this same method of duplication.*

Sorting A Portal At The Click Of A Mouse – Without A Million Layouts

On the CD-ROM: In the Sort Portal folder in the Chapter 5 folder.

Primary Database: SortPortal.fp5

The Problem: You want a portal to be able to sort by multiple fields that appear in the portal. (Click First Name and the portal sorts by first name; click Last Name and it sorts by last name; and so on.) The problem is that a portal's definition only allows for sorting by one set of criteria at a time. The only solution you can think of is to create a different layout for every sort order you want to see (click First Name and you're taken to a layout where the portal's defined to sort that way, and so on). But it's cumbersome to work with so many layouts and hard to build and maintain. Also, you would like to see something much more dynamic.

The Solution: Create a portal that sorts on a calculated field whose values vary depending on what a user selects in a global field. (See the solution database for more information on this stellar technique.)

The Hidden Portal Technique

On the CD-ROM: In the Hidden Portal Folder of the Chapter 5 folder.

Primary Database: Launch.fp5

The Problem: Sometimes you want buttons or other interface elements to only appear or function when a certain condition is met, only for certain users, or only when there is certain data in certain fields in a database. There are a few ways to do this, like creating a calculated container field that contains the graphic of a button that only appears when the criteria is met, but then the but-

ton still acts like an invisible button even when the criteria is not met. You could also just put the button there statically and control what happens in the attached script.

The Solution: Stack certain interface elements on top of a one-row portal. When the criteria you desire are met, this triggers a match field to populate, which then triggers this portal to become valid, which allows the interface elements stacked on top of it to become visible and functioning.

Launch the Launch.fp5 database and you'll be presented with the data entry screen for this solution, pre-built. Notice the question "Is This a Delivery?" at the top center of the screen, with a Yes/No drop down menu after it. On all new records, the default is no (this is a carry out order or whatever). Now, if you select "Yes" in the drop down and exit the record (by clicking anywhere else on the screen besides in a portal or in another field) you'll notice that a whole new box of information appears below the drop down menu with fields to fill in, having to do with shipment information. This is a hidden portal. So how is it done?

A hidden portal is basically a self-related portal on the screen that only appears (becomes valid) when a user enters some data into a field that makes the relationship valid. The trick centers around the fact that in FileMaker the contents of a portal will only appear if there is a valid relationship.

Why would you want to use a hidden portal? Well, often a developer will make a data entry layout so generic (to cover every possible variable in the ordering process) that the screen becomes cluttered with a whole lot of fields that Jane Doe user would rarely use on a standard order. This clutter makes data entry errors easier to make, causes the learning curve for using the database to go up, and makes the order data entry process, on average, slower. So why not only show groups of fields to the user if they are relevant? Enter the hidden portal!

Another reason why the hidden portal trick can be useful is in a database system where different users have different levels of access and some users need to see more options or buttons than others. Rather than building a separate layout, you can use the hidden portal trick to display those options or buttons (in the form of layout objects) only when the user has the appropriate access.

One downside to the hidden portal trick can occur if you place buttons in the portal row. If the button is set to change the pointer to a hand when it is over the button it will change to the hand even when there is no valid relationship for the portal. Clicking will have no effect except to confuse your users. The solution is to not have the pointer change to a hand, keep buttons out of the hidden portal, or use a separate layout instead.

A List View That's Actually a Portal

On the CD-ROM: In the Sort Portal folder of the Chapter 5 folder.

Primary Database: SortPortal.fp5

The Problem: You have a database with layouts designed to be an exact size, perhaps a kiosk solution. You want users to be able to see a list of all records in the database, or just certain records, but you want to keep the list view the same size as the detail view (or as the database window size-wise, anyway). You also want to add buttons or other layout objects to the side of your list view.

The Solution: Create a layout where the database shows a portal into a constant relationship to itself, sorted in an intuitive way. Now, you can see all the records in a list view-like format but it's contained in the neat, controllable format of a portal instead of the less controllable, larger, list type layout.

NOTE *Since this technique was already in use in the Sort Portal solution, we've referenced that same example database to explicate this concept.*

Highlighting Current Record in a List View Portal

On the CD-ROM: In the Highlight Current Record in the Portal folder of the Chapter 5 folder.

Primary Database: HighlightRow.fp5

The Problem: You have a self-related portal on a layout showing all records in the current database with some match to the current record (all records with the same last name, say). When using on screen navigation buttons you want the current record highlighted in the portal in yellow.

The Solution: Use a portal to a relationship based on that key field (LastName) and use button objects attached to a simple script to highlight the record.

Use a Multi-Keyed Portal to Quickly Add Items to A Related Database

On the CD-ROM: In the Conditional Portal Multikey folder in the Chapter 5 folder.

Primary Database: Launch.fp5

The Problem: You want a portal on one half of a layout that you can click into to add items to an order which is in a portal on the other half of the screen. You want to avoid using drop down lists or jumping between databases.

The Solution: Launch the Launch.fp5 database and look at the two portals at the bottom of the screen. The one on the left is a look into the products database (the stuff people can order from you). The portal on the right is a look into the line items database (stuff on this order). Above both portals are five global fields, cleverly labeled Type and Category1 through Category4. If you click on Type and select one, you'll see the product portal change. If you continue to select catego-

ries from left to right, the portal will show you a smaller and more specific list of your products, based on the selections that you make. How is it done? With a multikey portal.

What is a *multikey*? Basically, it is when you have more than one key per record on one side of a relationship and one or more keys per record on the other side of a relationship. In FileMaker, a multikey field is a field with more than one line of data in it, separated by carriage returns.

Setting It Up

The "key" to this multikey solution lies in the global fields mentioned above, the Type and Category fields. As you make selections in these fields, a hidden, unstored calculated field (text result), cCategories, is being built. cCategories is defined as:

```
gType & gCategory1 & gCategory2 & gCategory3 & gCategory4
```

As you can see, the field is simply a concatenation of the type and category fields all squeezed together. Once you've made a selection, the result of cCategories might look like this:

```
CatFoodSoupHot
```

The Products database also contains a field called cCategories (stored, text result), but this one is defined a little differently:

```
"All" & "¶" &
Type & "¶" &
Type & Category1 & "¶" &
Type & Category1 & Category2 & "¶" &
Type & Category1 & Category2 & Category3 & "¶" &
Type & Category1 & Category2 & Category3 & Category4
```

This field is setting itself up to be a multikey. If you select just a type and Category1 back in products, then the calculated line in the product database's cCategories field (`Type & Category1 & "¶"`) will allow you to see all records that match just that Type and Category1. If you make selections all the way down to Category4 in Invoice.fp5, you'll only match and see this line in the products portal in Products.fp5:

```
Type & Category1 & Category2 & Category3 & Category4
```

NOTE *Another cool feature that you could add to a solution like this is the same kind of multikey but for the line items portal. That way, you could see all products on the left, but only the, for instance, Category3 products on this order on the right. In other words, the portals into Products.fp5 and LineItems.fp5 would work independently of one another and be just as selectable by category. (You would have to make a second set of category globals and a second cCategories field, first, obviously.)*

Notice that there are little trash cans next to each category global field. This is to clear out the categories to select a new one or step back one level in the product categorization tree. Also notice that if you select, in this case, a home shopping (HS)

product in the Type field and then select down to Category3 and then go back and reselect Catering (Cat) Type, you'll see no products in the product portal. Why? Because there are no, for instance,

```
HSVeal
```

products, only

```
CatVeal
```

products.

To clear out all categories, you would have the user click the furthest left trash can which sets the portal to show *all* products. Or you could use a plugin like Script Scheduler to automatically refresh the categories if you go back and accidentally select a Type/Category combination that has no products.

Use Smart Scripting to Easily Add Items to an Order Without Using the Keyboard

Take a look at the scripts that allow a user to add items to an order. First, how does someone add an item to an order? Once you've selected the categories you want, simply click the items's name to add it to the order. Notice that, on a blank, new invoice, the item is instantly transferred to the first line of the right portal (LineItems.fp5). Also notice that the quantity defaults to one and that the line total and order subtotal auto-calculate. Now, if you click another item, it will add that to the order as well, on line two.

Now try to add another of the *first* item you selected. While you might expect FileMaker to add a third line to your line items portal, it doesn't. It simply ups the quantity of the already added item by one. Neat, yes?

Setting It Up

To set this up, you'll need to create a script in line items first called Increase Quantity. This is for the part of the script that ups the quantity if the item's already on the order. The script looks like:

```
Set Field [“Quantity”, “Quantity+1”]
```

Now go into Invoice.fp5 and create a script called Add Item to Order. Here's what the script is doing:

```
Freeze Window
#Sets a global to the ProductID to get product information without leaving this db via
a relationship.
Set Field [“gProductID”, “Products_ by cCategories::ProductID”]
Set Field [“gPrice”, “Products_ by gProductID to Product ID::Price”]
Set Field [“gInvoiceID”, “InvoiceID”]
#This is what it does if the product's already on the order.
If [“IsValid(Lineitm_ by cgInvoiceID\gProductID::ProductID)”]
  Go to Related Record [Show, “Lineitm_ by cgInvoiceID\gProd...”]
  #This is where it calls the Increase Qty script in LineItems.fp5.
  Perform Script [Sub-scripts, External: “Lineitm_.fp5 (209.187.70.46)”]
Else
```

```
  #This is what it does if the product's not on the order yet.
  Go to Field ["Line Items by InvoiceID::Line..."]
  Go to Portal Row [Select, Last]
  Set Field ["Line Items by InvoiceID::Prod...", "gProductID"]
  Set Field ["Line Items by InvoiceID::Price", "gPrice"]
End If
Go to Field ["Line Items by InvoiceID::Quan..."]
Go to Portal Row [Select, First]
#This loop simply drops the user off in the portal row they just edited/created.
Loop
  Exit Loop If ["Line Items by InvoiceID::ProductID=gProductID"]
  Go to Portal Row [Select, Exit after last, Next]
  Go to Field [Select/perform, "Line Items by InvoiceID::Quan..."]
End Loop
```

In keeping with the "no keyboard" philosophy, you can also delete a line item by clicking the trash can icon at the right of the line item portal. You can also increment and decrement the quantity for an item by clicking the up and down arrows next to the Quantity field.

NOTE *When you click the Decrease Quantity button when the quantity is at one, FileMaker asks if you want to delete the line item. This prevents a user from leaving an item on an order with a quantity of zero.*

Stacking Portal Fields Redux

Many people don't realize that you can stack fields in a portal. This means that you can place a field into a portal and then place another field, a seemingly infinite number of fields in fact (bonus points to whoever finds out the limit), right on top of it! To some this may sound clever but others will wonder, "Why do it? If one field's under another and you can't see it, then it's worth dry dirt to me!"

Here's a good reason to put stacked fields in a portal: limited space. Here's another good reason: don't force users to see every possible bit of data available all at once. Give them the option to get more data on the screen if they want, without leaving the current layout.

Try it! In Invoice.fp5, click on the little down arrow at the far right of the Products portal (the left portal). You'll notice that, for most products, you are taken to a hidden field in the portal that shows you the full description of a product that you sell. There's no reason to clutter up the portal with this field; it's best hidden until a user *wants* to see it.

Setting It Up

Go into your Products.fp5 database and create the Description field, a stored text field. Next create a calculated field called cDescription (text result), defined as:

```
Description
```

That's not a typo! This field is defined to be exactly what is in the Description field.

Now go back into Invoice.fp5 and add the related Description field to the products portal. Arrange the field to be just beneath the product name field. Now add a little arrow icon to the portal row and assign a script to it called Go to Product Description Field, defined as:

```
Go to Field ["Products_ by cCategories::cDescription"]
```

Now when you click on the little arrow, you'll be taken to the description for that product. And since you are viewing a calculated description instead of the original editable text field, a user won't be able to edit the master product description but can use it to get more info about a product for a customer on the fly.

Select A Portal Row To Data-Enter A Line Item

On the CD-ROM: In the Stack Select Dupe folder of the Chapter 5 folder.

Primary Database: Solution.fp5, layout TWO.

The Problem: A customer makes and sells all kinds of pumps. On any given invoice, a customer orders ten pumps at a time, on average. For each pump, there are 30 fields worth of customer-specific data that needs to be collected to customize each pump to suit the customer's needs (some pumps pump sulfuric acid, some pump poop).

How to enter this data into a FileMaker database? Well, most in-house developers create the invoice form layout and then just make a huge portal with gigantic rows, each of which is made to accommodate 40 plus fields! Your life really starts to suck as a data entry clerk, especially with large orders, as you scan the portals, field labels, and fields, trying to sort through the information.

The Solution: A better way to do this is to use a screen with a portal, through which you create, duplicate, and delete line items. Data is entered by creating or selecting an item in the portal which then populates fields on the form view outside the portal in an organized way. (Please see layout Two of the example database to get an idea of what we're talking about.) Now, all you have to do is press the "New Item" button, and a new item, highlighted in your list, appears. Then you type in the fields, as many as can fit on the whole screen, outside the portal without all kinds of crazy scrolling up and down a portal or right and left on a list view.

Setting It Up

1. First, in your line items database, create the following fields:
    - Item ID, an auto entered serial number
    - Constant, a calculated field, the calc being 1
    - g Highlight, a global container field that you've dropped a big block of highlight color into
    - g Item ID, a global number field
    - c highlight, a calculated container field defined as: `If(Item ID = g Item ID, g highlight,"")`

2. Now duplicate the two simple scripts, New Item and Delete Item, as they exist in the Support2 database, which is your Line Item database. Also, while you're in your line item database, create a quick relationship to the invoice database, called Invoices by Constant, using the Constant fields in both databases. Then go back to your invoice database and create the following fields:

    - g Invoice ID, a global number field
    - g Item ID, a global number field

    Now, create some relationships in the Invoice database:

    - Line Items by Invoice ID (allow creation of related records)
    - Line Items by Constant
    - Line Items by g Line Item ID to Line Item ID

    Now you can put a simple portal on your form view using the "by Invoice ID" relationship, showing about ten rows. Put just the related Item [name] field in the portal and disallow entry to it; also, make it transparent.

3. Place the related c Highlight field right beneath the Item field. Place all of your line item's fields to the right, not in the portal, fields like Item, Quantity, Color, Price using the "by g Line item ID to Line Item ID" relationship.
4. Create two scripts in the invoices database called "New Line Item" and "Select Line Item." These should look just like the ones in your SOLUTION.FP5 database.
5. Create a button on the layout called New Item and associate the "New Line Item" script to it. Make the c Highlight field a button and associate the "Select Line Item" script to it. Now when you click on an existing item, it will populate all the fields to the right so you can enter and edit them. When you click "New Item", you can do the same with a new line item. You can also add a "Delete Line Item" button and scripts to the databases as in the example (click the little trash can).

That's it! Now your users are ready to start adding items in an organized, pleasant-to-look-at screen.

The only drawback with this method of creating line items in Invoices or Purchase Orders is that you can't see all the items on one layout at the same time. You could always create another display list layout, though.

One other possible drawback to this solution is that if you don't rely on on-screen buttons for navigation through these databases, it's possible for a user to be looking at invoice 1001 with the line items for invoice 3001. See, when you fill in "g Line Item ID" in the Invoice database (the match field that tells you which line item to view to the right), it stays set until it is reset by your "Select Item" button. This happens even if the user switches records using the book on the status bar, the keyboard, or the mouse wheel. You don't want this to happen because it will confuse and befuddle people. Plan for it!

Rotating a Portal

On the CD-ROM: In the Rotated Portal folder of the Chapter 5 folder.

Primary Database: Invoice.fp5

The Problem: You want a portal to be rotated sideways.

The Solution: This is sort of a trick, because this solution does not actually include the use of a portal since portals cannot be rotated in FileMaker Pro. However, with the use of a few extra relationships, you can simulate the look and feel of a rotated portal, provided that the number of related records is won't be a large number.

Other Cool Portal Tricks

There are many more cool things you can do with portals, too many to cover in a book such as this. But here are a couple that you'll find handy to have in your bag of tricks.

The "Viewer Database" Development Method

Some FileMaker developers really dig what's called the "Viewer database" (also known as "thin client") method of FileMaker design whereby you make an empty, record-less, FileMaker-built front-end to a back end FileMaker database system. In other words, you separate a database system's structure and interface from its data, which is what most other big RDMSs do (like a Visual Basic or Web-based front-end to a back-end Microsoft SQL Server 2000-built database system). Usually with this method you have a one-record Menu or "Interface" database that sits on each client computer and points to the "real" databases which live on a FileMaker Server somewhere on the network.

The great thing about using the viewer file method is that all navigation, scripts, and other structure is centralized in one database, allowing you to more securely control the modification of data. (The front end is easily installed on any new workstation via drag and drop.)

The downside is that your front-end database can get quite large and somewhat unwieldy to develop. It may also require more development time since all back-end database records are now created, updated, and deleted using scripts, multikey portals, and complex relationships in the "viewer database".

NOTE *To learn more about this advanced method of FileMaker database development, contact Colleen at Datawaves (see http://www.data-waves.com/) who recently led a session on the topic at FileMaker Developer Conference 2002.*

The Hierarchical Portal

Sometimes you want to create a portal that works kind of like Windows Explorer or the Mac's Finder (pre OS X, that is), allowing you to "turn down" portal rows to see their "children" in the same portal, and even drag portal rows into other rows in the same portal to "nest" them within a hierarchy. To learn how to do this, download the

free fully unlocked example file from the Cleveland Consulting Web site at http://www.clevelandconsulting.com. (Click the Materials & Support link, then click Downloads, Tools & Whitepapers. The link to download the Hierarchical Portals.fp5 example database is at the bottom of the new page that appears.)

NOTE *This Cleveland Consulting example requires the ScriptScheduler and Troi Text plug-ins but demo versions of each are included in the download.*

Planning a Relational Database System

One of the hardest things for new relational database developers to grasp sometimes is when to break off some bits of data from one database into another related database. For example, in a contacts database, any given contact might have up to three phone numbers, but usually not many more than three. So you ask yourself, do I need a related database hanging off of Contact.fp5 called Phone.fp5 to keep track of all the contacts' related phone numbers using good database form?

The answer to this question is always going to be up to you. Plan your database well before you even start, determining how each database will relate to each other and what databases are required.

When deciding whether to split some data off to another related database, ask yourself:

- How many "repeats" of this bit of data will each master record need at most? (Two, three, or four phone numbers per contact? Or more than ten on average?)
- How many databases are already part of this solution and can I afford to add another? (FileMaker Pro and FileMaker Server have upward limits of how many databases they can serve at one time.)
- Can the data I want to break off be kept track of in an existing database? For instance, you may already have a database for client phone numbers that can be modified a bit to keep track of client, vendor, employee *and* personal phone numbers.
- Do I need to do any reporting on or summarizing of the data in question? (If so, this usually indicates the need for a related database.)

If you still aren't perfectly sure what to do, build a little model of what you need outside the main system and test it. Draw an ER diagram (more on this in Chapter 15) so that you can visualize how everything relates to everything else.

The Last Word

In this Chapter, we've unraveled the mystery of relational database systems and you learned how to set them up using FileMaker Pro. You've also learned handy techniques to power up your solutions that are *not* covered in the FileMaker Pro manual. Use them wisely! Don't add them in if they don't answer a specific, important business need. Less is often more as the saying goes.

In Chapter 6 we'll look at another one of the tools in our tool chest of powerful FileMaker development: calculation formulas — fields that calculate a result based on an almost infinite number of variables that you define.