

# Deleting Records, Reports, Data Design, and Reader Contributions

Peter Vogel



Peter Vogel answers a variety of questions centered around deleting records in subforms, setting the RecordSource dynamically in a report, and modeling data. He also gets some help from a reader and tells another to “Just Say No.”

I have a master/detail form. I want to have a button on the main form that deletes the current record in the subform. That is, if the user clicks on a record in the subform and then clicks the Delete button on the main form, the record the user selected is deleted.

I don't have a sample of your database, but one suggestion that you might want to consider is to put the Delete button on the subform. I'm concerned that if you put the Delete button on the main form, the user may think that clicking the button will delete the master record and its children, rather than just the currently selected child. One solution is, of course, to have two Delete buttons (one for the master record and one for the children), but if you've done that it seems to me that the right place for the Delete Child button is on the subform.

After that unsolicited advice, let's look at your problem. The first thing that you need to do is gain access to the properties of the form with your detail records. If you created the subform by dragging a form onto your main form, Access will have done three things for you:

1. Created a subform control on your main form with the same name as your subform.
2. Shoved the subform into that control.
3. Set up the subform's Master/Child properties to keep the main form and subform synchronized.

So, to get to the subform, you must go to the subform control that holds it (which has the same name as the subform) and then use the Form property of the subform control to gain access to the form. If your subform was called frmOrders, then the subform control that holds it is also called frmOrders. To retrieve the subform's caption, you'd use:

```
Me.frmOrders.Form.Caption
```

Now let's look at deleting the record. There's probably a better way to do this, but here's how I've handled the problem in the past. The first thing that you need to know is that the form knows which record is selected. One place that you can retrieve that information is through the form's Bookmark property (another is the CurrentRecord property). The Bookmark property returns an internal string that uniquely identifies the record (the CurrentRecord property returns the position in the recordset of the current record). Unfortunately, the value in the Bookmark property can only be used in the recordset that the form is based on. Even if you issued a query that returned exactly the same records, you wouldn't be able to use the form's Bookmark.

Fortunately, the form's RecordsetClone property does give you access to the recordset that the form is based on. All you need to do, then, is move the RecordsetClone to the same record that the form is on by setting the RecordsetClone's Bookmark to the form's Bookmark. You can then call the Delete method to remove the record, like this:

```
Me.frmOrders.Form.RecordsetClone.Bookmark = _
    Me.frmOrders.Form.Bookmark
Me.frmOrders.Form.RecordsetClone.Delete
```

You can't just use the Delete method with the RecordsetClone, unfortunately. The current record recordset for the RecordsetClone property doesn't necessarily match the current record for the form. Calling the Delete method of the RecordsetClone without setting the current record will just delete the first record in the recordset.

One other comment: You can make your code run a little bit faster by setting a variable to point to Me.frmOrders.Form. That way, VBA doesn't have to keep searching through the objects to get to the one that you actually want (or, more accurately, does the search only once when you set the variable). In this example, I've also set a variable to point to the recordset:

```
Dim frm As Form
```

```
Dim rst As DAO.Recordset

Set frm = Me.frmOrders.Form
Set rst = frm.RecordsetClone
rst.Bookmark = frm.Bookmark
rst.Delete
```

I should point out that you shouldn't expect a tremendous speed gain here: We're talking about saving milliseconds, not seconds.

If you're going to set a variable to point to the recordset, you must make sure that you pick up the right object library. The RecordsetClone property returns a DAO recordset (not an ADO recordset). In addition, you must make sure that you have the right version of DAO checked off in your References list. For instance, with Access 2002, you must check off Microsoft Data Access Objects 3.6 in order for the line that sets the rst variable to work.

This technique works, by the way, even if your subform is a datasheet. The sample in the Download file (available at [www.vb123.com/kb](http://www.vb123.com/kb)) uses a datasheet in the subform.

[I have a form whose RecordSource is set from a variety of parameters. Once the form comes up, I want to let the user print the data using one of several different reports that I've created. Because the SQL statement in the RecordSource isn't fixed, I'd like to be able to set the RecordSource in my report from the RecordSource in my form.](#)

Once again, I'm sure that there's a better way to do this than the mechanism I'm going to suggest. Among other things, this method shifts nervously back and forth between two different ways of working with reports. It also seems to involve a certain amount of magic.

The first thing that you need to do is open the report so that you can make changes to it. However, to change the RecordSource, you must open the report in design view. To do that, I use the OpenReport method of the DoCmd object:

```
DoCmd.OpenReport "rptCustomers", acViewDesign
```

I'm always suspicious of any method that depends on using DoCmd because the object started life as a command to be used in Access macros.

Now that the form is open, you can retrieve it from the Application object's Reports collection. Once you've retrieved the report, you can change the report's RecordSource:

```
Application.Reports("rptCustomers").RecordSource = _
    Me.RecordSource
```

Since I'm only going to use the report once, I don't bother setting a variable to point to the report. The cost of setting the variable to point to the report is the same as

retrieving the report to set its RecordSource, so nothing would be gained.

Now you need to either have the report print or, if you want your users to have a look at it before it goes to the printer, display in print preview. This is where the magic comes in because I use the DoCmd's OpenReport method to open the Report again in whichever mode I pick (acViewNormal to print or acViewPreview for print preview). Rather than open a new copy of the report, Access changes the mode of the report that's currently open:

```
DoCmd.OpenReport "rptCustomers", acViewNormal
```

or

```
DoCmd.OpenReport "rptCustomers", acViewPreview
```

If you've switched the report to acViewNormal to print it, you'll probably want to close it. If you don't, the report stays on the screen in preview mode until your user closes it—at which point they'll get a message asking them if they want to save the changes to the report. The code to close the report uses the DoCmd object again and, again, DoCmd magically works with the currently open report:

```
DoCmd.Close acReport, "rptCustomers", acSaveNo
```

I use the acSaveNo option so that the report will be saved without the updated RecordSource. If you create the report with its RecordSource set to some default SQL statement, users can run the report from the database window without getting an error message. By not saving the change to the RecordSource, that default statement won't be overwritten.

If you do switch the report to acViewPreview, *don't* close the report from your code. If you do, all the user will see is a brief flash on the screen as your code opens and closes the report. The sample database in the Download file sends a boring six-page report to the printer, so you may want to alter the code before running it.

[I have a set of text boxes that the users must enter text into before saving their record. How can I prevent them from leaving each text box until they've filled it in?](#)

This is a good question and I've got an offensive answer: Don't. Unless you have a very good reason for forcing the users to enter their data in a particular order, why shouldn't they enter the text boxes in any order that they want? I remember one system I saw that did force the user to enter data in a particular order. Working with one user, I noticed that she had written a series of entries on her desk pad. It turned out that, on occasion, she was assembling the data in an order different from the one

required by the system. To handle this, she would write all the information down on her desk pad until she had a complete set. Then she would enter the data into the form.

By “a very good reason” for controlling the data entry, I don’t mean, “I believe (sitting here at my desk writing the code and not actually having to do my users’ job) that it’s more efficient to enter the data in this order.” If the order that you want to impose really is more efficient, your users will figure that out for themselves. In the meantime, just because you can’t imagine a scenario when the user would want to enter the data in some different order, it doesn’t mean that such a scenario can’t occur.

The only reason that I can think of for forcing the user to enter the data in a specific order is because data in the first text box is used to prompt the user on what to enter in the second text box *and* it’s unlikely that the user will get the entry in the second text box right without your prompt. Even if the data in the first text box is used to validate data in the second text box, I wouldn’t recommend forcing the user into a particular order. You can still cross-validate the data before saving without doing it on a text box by text box basis. Only if you’re going to provide some positive guidance for entering the data correctly should you spend the time to enforce the order of the data entry—and even then, I’d only do it if I expected my users to need my help (if, say, they’re getting it wrong more than 10 percent of the time).

Just to be clear: I’m not suggesting that you don’t want to edit your data before saving it to make sure that all the required fields are filled in.

[We’ve got an argument going on at my work about the best table design for a particular problem. We manage pensions for a number of clients. Most of these clients work for a small number of employers. This means that the contact address for many of our clients is the same—it’s the employer. Should we set up our database to store each client’s address separately, or should we just point all the clients for an employer to the same address?](#)

This is a tough question, and I’m not surprised that you’ve got disagreement about what the right answer is. I suspect that there will be some disagreement about my answer, too.

I believe that the purpose of your data design is to model the world that your business works with, as your business sees it. So, the first question that you need to ask is, what’s the right model for your business? For instance, do you look at your clients as separate individuals who, by some cosmic coincidence, all have the same employer? Or do you look at your clients as groups organized by employer? If you answered the first question with a

“Yes,” then I’d store each address separately. If you answered the second question with a “Yes,” then I’d allow clients to share addresses.

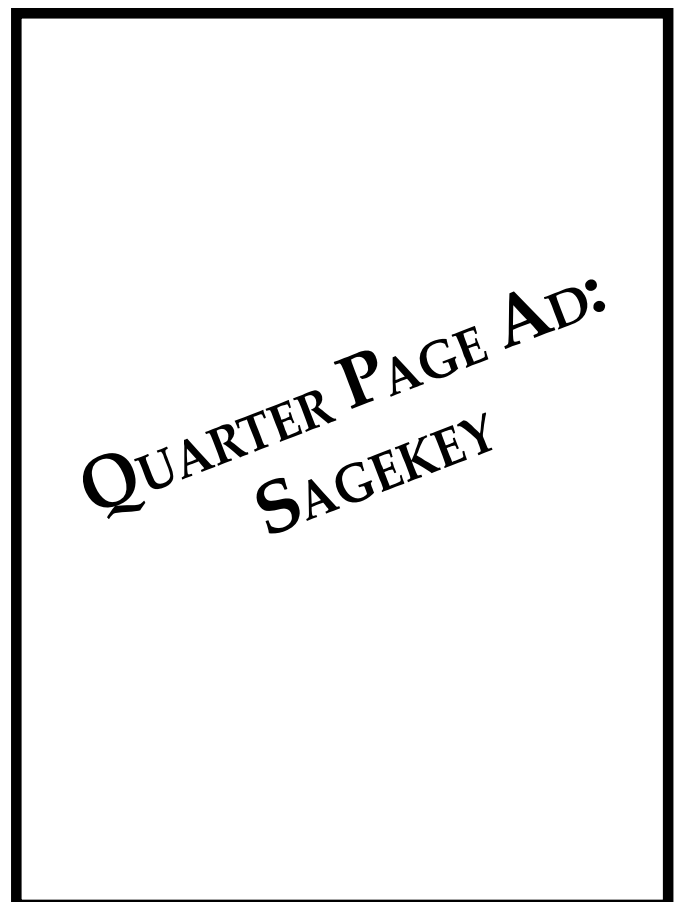
The second answer can give you the best of both worlds. If you put the addresses in a separate table with a unique identifier, then you can use a foreign key in the employee table to link employees to addresses. Your user interface would have to give you the ability to select an address from the table or to add a whole new address.

To support assigning common addresses, you’ll need some way to flag the “sharable addresses.” One way to do that, of course, is to add a “sharable” flag to your address database. However, if the only shared addresses are those belonging to employers, then I’d use that relationship to find the sharable addresses. This would mean either having a table of employers that link to the address table or (if your employers are just another client in your clients table) flagging the employers in your client table.

### Feedback

A few issues back, I discussed how to create a form that would expand to reveal more of itself when you clicked on an “Advanced” or “Details” button (or just “>>”). SFC Gregory Andrews of the Northeast Information Operations Center at the Devens RFTA, in Massachusetts,

*Continues on page 19*



## Access Answers...

*Continued from page 17*

sent a follow up e-mail with this suggestion:

“I was recently reading your ‘Access Answers’ column on expanding and contracting forms in the December 2002 issue of *Smart Access*. After contracting the form, you noted that ‘the user can still get to the controls in the lower (hidden) part of the form by tabbing to them.’ Which is correct.

“But I’ve used this same method to expand and contract forms in some of my applications, and I’ve come up with a solution that doesn’t allow users to tab to the controls in the lower part: I use a transparent command button. For the `OnGotFocus` property of this transparent command button, I set the focus to another control on the form—usually the first control on the form. I adjust the

tab order of my form so that my transparent command button is after the last control on the upper part of the form in the tab order. When users tab through the form, they arrive at the transparent command button, and are rerouted to the first control on the form, bypassing the hidden fields.

“When the user clicks on the command button to expand the form, I make the transparent command button invisible, which removes it from the tab order. The user can now tab through to the lower half of the form. When the user contracts the form, I make the first command button visible (though still transparent).

“This is certainly easier than hiding and revealing every control on the lower part of the form. I’ve gotten so much out of *Smart Access* over the years I just wanted to

add my two cents.” ▲

**DOWNLOAD** [AA0403.ZIP at www.vb123.com/kb](http://www.vb123.com/kb/AA0403.ZIP)

Peter Vogel (MBA, MCSD) is the editor of *Smart Access* and a principal in PH&V Information Services. PH&V specializes in design and development for systems that use Microsoft tools. Peter has designed, built, and installed intranet and component-based systems for Bayer AG, Exxon, Christie Digital, and the Canadian Imperial Bank of Commerce. He also wrote *The Visual Basic Object and Component Handbook*. In addition to teaching for Learning Tree International, Peter wrote its Web application development, ASP.NET, and technical writing courses, along with being technical editor of its COM+ course. His articles have appeared in every major magazine devoted to VB-based development, can be found in the Microsoft Developer Network libraries, and are included in Visual Studio .NET. Peter also presents at conferences around the world.

## April 2003 Downloads

- [WORKFLOW.ZIP](#)—Mike Gunderloy has provided the sample files that he describes in his article on using the Workflow Designer. This includes the Access Data Project that manages the SQL Server files, the detached SQL Server files with the tables that Mike used, and the Data Access Project that generated the Web pages used in the simplified workflow for Engineering Change Orders that Mike developed.
- [CRREPOS.ZIP](#)—For his article on integrating the Crystal

Reports Repository with Access, Gord Maric has provided sample repository database for you to examine. All of the sample items discussed in Gord’s article (SQL statements, text, and functions) are in this database.

- [AA0403.ZIP](#)—Peter Vogel has included the sample code from his “Access Answers” column for this month. Sample code shows how to delete a record in a subform from the main form and setting a report’s RecordSource from a form.

[www.vb123.com/kb](http://www.vb123.com/kb) with your unique subscriber user name and password. For access to this issue’s Downloads only, click on the “Source Code” button, select the file(s) you want from this issue, and enter the User name and Password at right when prompted.

User name

Password

Editor: Peter Vogel  
 Contributing Editors: Mike Gunderloy, Danny J. Lesandrini, Garry Robinson, Russell Sinclair  
 CEO & Publisher: Mark Ragan Group  
 Publisher: Connie Austin Executive  
 Editor: Farion Grove Production  
 Editor: Andrew McMillan

### Questions?

Customer Service:  
 Phone: 800-493-4867 x.4209 or 312-960-4100  
 Fax: 312-960-4106

### Subscription rates

United States: One year (12 issues): \$169; two years (24 issues): \$287  
 Other:\* One year: \$194; two years: \$330

Single issue rate:  
 \$20 (\$25 outside United States)\*

\* Funds must be in U.S. currency.

*Smart Access* (ISSN 1066-7911)  
 is published monthly (12 times per year) by:

Pinnacle  
 A division of Lawrence Ragan Communications, Inc.  
 316 N. Michigan Ave., Suite 400  
 Chicago, IL 60601

POSTMASTER: Send address changes to Lawrence Ragan Communications, Inc., 316 N. Michigan Ave., Suite 400, Chicago, IL 60601.

Copyright © 2003 by Lawrence Ragan Communications, Inc. All rights reserved. No part of this periodical may be used or reproduced in any fashion whatsoever (except in the case of brief quotations embodied in critical articles and reviews) without the prior written consent of Lawrence Ragan Communications, Inc. Printed in the United States of America.

Brand and product names are trademarks or registered trademarks of their respective holders. Microsoft is a registered trademark of Microsoft Corporation. Microsoft Access is a registered trademark of Microsoft Corporation. *Smart Access* is an independent publication with Microsoft Corporation. Microsoft Corporation is not responsible in any way for the editorial policy or other contents of the publication.

This publication is intended as a general guide. It covers a highly technical and complex subject and should not be used for making decisions concerning specific products or applications. This publication is sold as is, without warranty of any kind, either express or implied, respecting the contents of this publication, including but not limited to implied warranties for the publication, quality, performance, merchantability, or fitness for any particular purpose. Lawrence Ragan Communications, Inc. shall not be liable to the purchaser or any other person or entity with respect to any liability, loss, or damage caused or alleged to be caused directly or indirectly by this publication. Articles published in *Smart Access* do not necessarily reflect the viewpoint of Lawrence Ragan Communications, Inc. Inclusion of advertising inserts does not constitute an endorsement by Lawrence Ragan Communications, Inc., or *Smart Access*.