

Smart Access

Solutions for Microsoft® Access® Developers

Temporary Tables with No Bloat

Doug Den Hoed



Temporary tables are great for extending query functionality, storing transient data, and solving concurrency issues. Unfortunately, using a linked Jet table to store that data will cause database bloat. This month, Doug Den Hoed shares his technique for creating bloatless CSV temp tables on the fly.

SOMETIMES you need to hold some data in a table to work with it over the course of a transaction, but once the transaction is complete the table can be discarded. This is the essence of temporary tables. I've taken advantage of temporary tables in many of my applications. Unlike SQL Server and Oracle, however, the Jet engine doesn't directly support the concept of temporary tables. To make things worse, even using a standard table as a holding area and deleting the records after processing doesn't work very well in Access. Jet doesn't recover the space from deleted records until the MDB is compacted, a behavior Access developers call "database bloat." A Jet application that uses anything like temporary tables will just grow and grow until it consumes the user's hard disk.

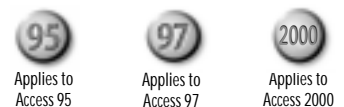
I tried a number of ways to use temporary tables efficiently in Access. At one point I was writing code to open a new MDB, create a table in it, link the tables back to my original MDB, insert the temporary data, use the record, and then delete the MDB. No luck: Bloat occurred anyway in my source MDB. I was also running into concurrency problems. If two users were running a procedure that created a temporary table, I needed some way to keep their data separate. "Temporarily" defeated, I explained to my users that they'd need to compact their databases often, and I even created compaction routines for my users to run. Then I found a solution.

Continues on page 5

January 2001

Volume 9, Number 1

- 1** Temporary Tables with No Bloat
Doug Den Hoed
- 4** Editorial: Access Applications
Peter Vogel
- 10** User-Driven Menus
Keith Bombard
- 14** Everything Doesn't Happen at Once: Loosely Coupled Events
Peter Vogel
- 17** Access Answers: Data Projects and SQL Server
Andy Baron
- 22** January 2001 Source Code



www.vb123.com/smart

In code, an underscore (_) as the last character of a line indicates that the line has been wrapped for layout purposes. In Access 95 and up you can use the code as it appears, but in Access 2.0 you must recombine the wrapped lines.

Temporary Tables...

Continued from page 1

Needing temporary help

I was compelled to find a better way while preparing to release a commercial application called The KB™. The KB is knowledge base (“KB”) software designed to manage team-oriented, task-driven projects—projects like building Access applications, for instance.

Certain charts in The KB let users dynamically sort a datasheet’s column headers (see my article in the May 2000 issue of *Smart Access*, “Taking Back the Power with the Access Runtime”) and then use that data to construct a Gantt chart in MS Graph (I’ll cover this in an upcoming article). The process makes extensive use of temporary tables, and I worried about the bloat from running those charts. Since we were creating a commercial product, I wondered how I could impose my “compact often” workaround on an anonymous, international audience. So, instead of storing my transient chart data in Jet, I devised a technique to write my temporary data to a local comma separated value (.CSV) file, link the CSV file, and then use the linked CSV in my queries like any other table. This technique turns out to be fast, causes no bloat, and solves concurrency issues since users each have their own local datasource for their charts. Do note that I use these tables only for reading data and don’t try to update them (which is generally all that’s required of a temporary table).

I ran some tests to benchmark these different bloat scenarios and summarized the results in [Figure 1](#). You can generate your own benchmarks using the sample database that’s available in this month’s Source Code file at www.smartaccessnewsletter.com. It contains a routine called `tmpCreateFromListbox` that can be added to your applications to send criteria from a multi-select list box to a CSV file and automatically link the file back to your database to be used in SQL queries. I’ll walk through that function to show off the technology, and then I’ll give you some examples of ways I’ve applied the TempCSV technique.

The sample database’s “skills application” illustrates how my function could be used. The sample is based on a real application that I built for one of my clients. The

original solution used a lot of code but didn’t require any temporary tables. By applying the techniques that I’ll show you here, I rewrote the solution to use SQL and considerably less code.

The problem that the sample addresses is to find a consultant who meets a specific set of criteria. The criteria can be divided into three categories (industry, skills, and language), and the user is allowed to specify multiple values in each category. In [Figure 2](#) (on page 6), you can see the main form from the sample database at the top of the figure and the query that retrieves the results at the bottom. In the form’s top three list boxes, you can multi-select combinations of Industry (in this case, Oil & Gas), Skill (here, Access, ADO, DAO, SQL Server, and Visual Basic), and Language (in this case, <Any>) values. Each list box has two columns: the ID (hidden) and the value (shown). So, in my example, `ttmpIndustry.CSV` has 1 for Oil & Gas; `ttmpSkill.CSV` has 1, 4, 5, 6, and 2 for my five skill choices; and `ttmpLanguage.CSV` has 6, 5, 1, 2, 4, and 3, since I chose <Any>. Clicking one of the buttons on the right calls my `tmpCreateFromListbox` function, which creates temp CSV tables that contain just the IDs for the selected items.

The query (`qselQualifiedConsultant`) at the bottom of [Figure 2](#) uses the CSV tables as part of a SQL statement that retrieves all of the employees who meet the criteria selected in the list box (thanks to Peter Vogel for helping me work out the SQL for this). The CSV tables restrict the data to show only the consultants who match the values selected from the lists. The SQL statement joins each CSV

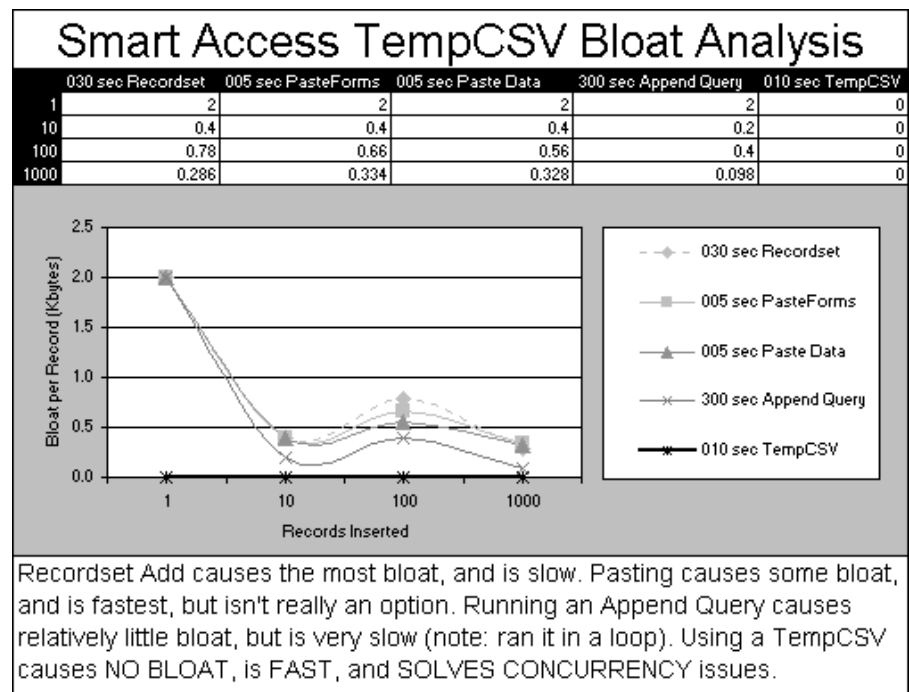


Figure 1. Bloat per row benchmarks.

table to the corresponding Xref table, which list the industries, skills, and languages for each consultant. Those tables are, in turn, linked to the list of consultants, which limits the rows to only those consultants who meet the criteria in all three tables. The Group By converts the records from the joins to a single row, which I then display in a message box, on a report, on the form, or on a subform, depending on which button you click to run the application. The sidebar “No More Compacting?” has a further explanation of the buttons on the form.

Linking tables

Unlike a table, a comma-delimited file doesn't contain a description of its structure. To link a CSV, you need to create a Link Specification so that Access knows how to parse the information stored in the CSV. The simplest way to build that specification is to link a text file to your database through the Access user interface. Since my tables were going to be generated as my application ran, I had to create a dummy file to link to Access in order to create the specification. My three CSV files could use the same specification since they had identical layouts: one field (datatype Long) per record. To create the dummy file, I opened a new file in Notepad, typed “1”, hit Enter, typed “2”, hit Enter, typed “3”, and then saved the file as ttmpLongID.txt. I now had a text file of three records, each record having one field, and Long data in each field.

In Access, I followed these steps to link in my dummy table and create my Link Specification:

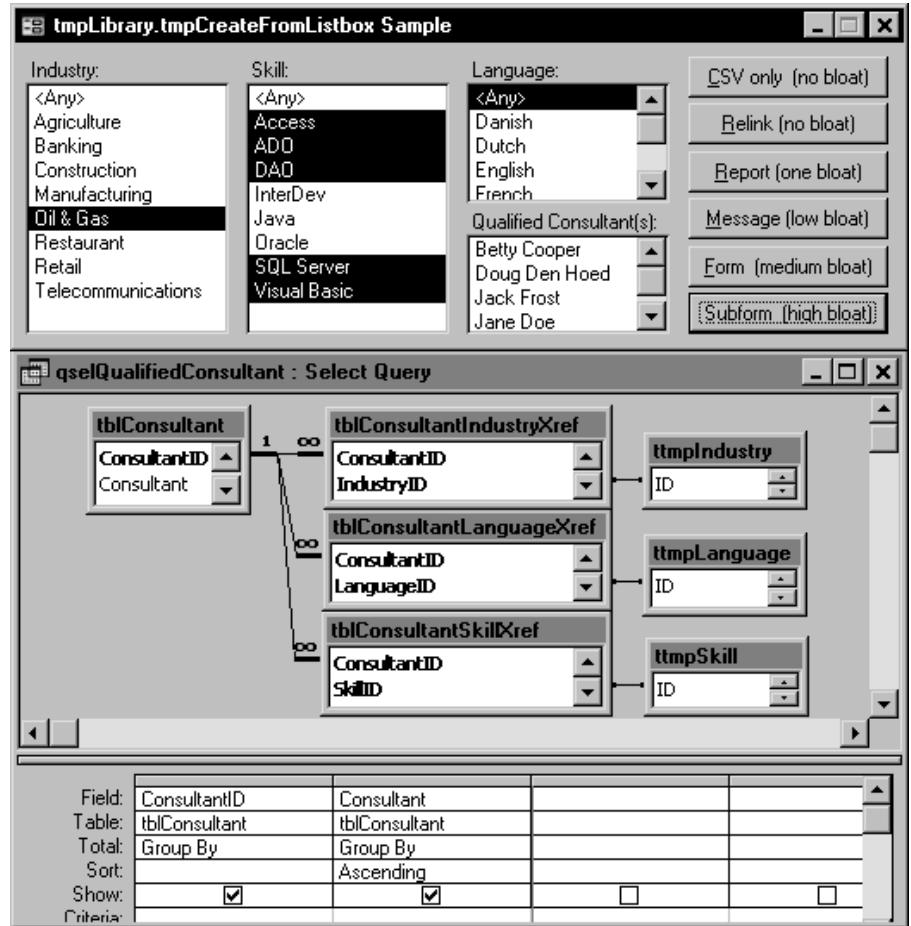


Figure 2. tmpCreateFromListbox example.

1. From the File menu, I selected Get External Data | Link Tables.
2. In the Link dialog box that appeared, I changed Files of Type to Text Files, selected the file ttmpLongID.txt, and clicked on the Link button.
3. In the first form of the Linked Text Wizard that appears next, I chose Delimited and clicked on the Next button.
4. The next form in the Wizard lets you select the

No More Compacting?

My sample database includes some extra buttons that demonstrate something I discovered while writing this article: Some very common Access actions cause database bloat. I found that, in both MDB and MDE formats, creating, loading, and linking CSV files causes no bloat. Running a report that refers to CSV tables sometimes adds about 8K across multiple runs. Walking a recordset to format a

MsgBox causes less, about 4K every time, on the OpenRecordset. Opening a bound form causes some bloat every time it opens (again, 8K). Changing the RowSource on a subform causes the most bloat every time (12K). Use my sample form to test the effect on your database, as you might get different results on your machine, but I suspect the general behavior is unavoidable.

character that you'll use to delimit the fields in your record. I selected Comma as delimiter and clicked on the Next button.

5. The form that appears next in the Wizard allows you to assign names and data types to the fields in the text file (the Wizard scans my dummy text file to determine how many fields are in it). I named the field "ID," set its data type to Long Integer, and clicked on the Next button.
6. The last form in the Wizard lets you assign a name to the table. I named my table Linked Table TtmpLongID and then clicked on the Finish button.

The Wizard has inserted a row in MSysIMXSpecs that describes my Link Specification, which I can now use to link to other tables with the same layout. The entry in MSysObjects for the link to my dummy table uses that specification in its connection string as "DSN=TtmpLongID Link Specification," for instance.

My tmpCreateFromListbox function accepts four parameters: the name of the table to link, the specification to use, the list box to draw the data from, and a force link flag. To save processing time, I normally only link the table if it's not already linked (for instance, the first time on a new computer). However, passing a True in the force link parameter to the call will cause the routine to drop any existing link to the table requested and create a new one. A call to link a file called ttmpIndustry using my predefined specification and load the table with data from the list box lstIndustry (without forcing a relink) would look like this:

```
Call tmpCreateFromListbox("ttmpIndustry", _
    "TtmpLongID Link Specification", _
    Me.lstIndustry, _
    False)
```

I use ttmp as a prefix for all of my temporary tables. I also give my tables the same name as the file that they're linked to. So, in the preceding example, I'm linking the file ttmpIndustry.CSV as the table ttmpIndustry.

Following is the start of the tmpCreateFromListbox function. The first thing that it does is check to make sure that at least one item in the list box is selected and exit if not:

```
Public Function tmpCreateFromListbox( _
    ByVal strTempTable As String, _
    ByVal strLinkSpec As String, _
    ByRef lstSource As ListBox, _
    Optional ByVal fForceRelink As Boolean = False _
    ) As Boolean

If lstSource.ItemsSelected.Count <= 0 Then
    GoTo Exit_tmpCreateFromListbox
End If
```

Providing there's at least one row selected in lstSource, tmpCreateFromListbox continues, initializing

the fFirstRow flag and strCSVFile location. The code also deletes any existing versions of the strCSVFile and then opens a new file for Output:

```
fFirstRow = True
strCSVFile = tmpApplicationPath & "\" _
    & strTempTable & ".CSV"
On Error Resume Next
Kill strCSVFile
On Error GoTo Err_tmpCreateFromListbox
lngFileNumber = FreeFile
Open strCSVFile For Output As #lngFileNumber
```

Next, the function determines whether the list box is disabled and checks for a special "*" value in the selected row. Either condition means that every value in the list box should be inserted to the TempCSV, a decision that's stored in the fDoAll variable:

```
fDoAll = False
If Not lstSource.Enabled Then
    fDoAll = True
Else
    For Each varItem _
        In lstSource.ItemsSelected
        If lstSource.ItemData(varItem) = "*" Then
            fDoAll = True
            Exit For
        End If
    Next varItem
End If
```

Next, the code writes the selected items in the list box to the TempCSVfile or, if fDoAll is set to True, every item:

```
If fDoAll Then
    varItem = 0
    Do Until varItem >= lstSource.ListCount
        If lstSource.ItemData(varItem) <> "*" Then
            Write #lngFileNumber, _
                CLng(lstSource.ItemData(varItem))
        End If
        varItem = varItem + 1
    Loop
Else
    For Each varItem In lstSource.ItemsSelected
        Write #lngFileNumber, _
            CLng(lstSource.ItemData(varItem))
    Next varItem
End If
```

With the file now loaded with the data, the function issues a DCount against the TempCSV to check the link. If the DCount fails, or the routine has been passed the force link parameter, the code deletes the current link and relinks using the name of the Link Specification passed to it:

```
On Error Resume Next
Close #lngFileNumber
Debug.Print DCount("...", strTempTable, "1=2")
If Err <> 0 Or _
    fForceRelink Then
    On Error GoTo Err_tmpCreateFromListbox
    On Error Resume Next
    DoCmd.DeleteObject acTable, strTempTable
    On Error GoTo Err_tmpCreateFromListbox
    DoCmd.TransferText acLinkDelim, _
```

```

strLinkSpec, strTempTable, _
strCSVFile, False
Else
  On Error GoTo Err_tmpCreateFromListbox
End If

```

On exit, the function will close the CSV file, no matter what else happened in the routine. Without that precaution, subsequent attempts to work with the file will fail. Similarly, any objects that refer to the TempCSV must also be closed before you can refresh them, or you'll get "Err 70: Permission Denied" when you try to open the file for Output. Once all of the temporary tables are linked, I run the query qselQualifiedConsultants that generates the list of consultants.

When you use tmpCreateFromListbox in your applications, remember that you can store any data you wish in the CSV file, not just Long values as I chose to do. Just remember to create the Link Specification first. You can choose a different delimiter, which is essential if the data you intend to write to the file has commas in it.

Extending query functionality

So how can you use these temporary tables? For my

first example, I'll take a user who enters some project information into The KB and then generates a chart to review the impact of those changes. In **Figure 3**, you can see our KB product in its docking station mode. In the top right subform, two Treeview controls allow the users to show and hide the data they're interested in (see my October 2000 article, "Taming the Treeview Control") to set the overall context for the data they'll work with. The user then refines the search in the top left subform by choosing The KB project, which in turn drives its parent, Software Development, into the template field. In this example, Doug Den Hoed has been chosen in the solver field.

When the user clicks the filter button, The KB will search for any "KB Items" that match the filter (in this case, the result corresponds to the question, "What is assigned to Doug Den Hoed in The KB project that isn't closed?"). The results are displayed in the bottom left pane. The user can scroll right to confirm that each KB Item has a budget, revised budget, and estimate to complete hours. Double-clicking the row for KB Item 923 pops up its detailed form. The user enters some actual

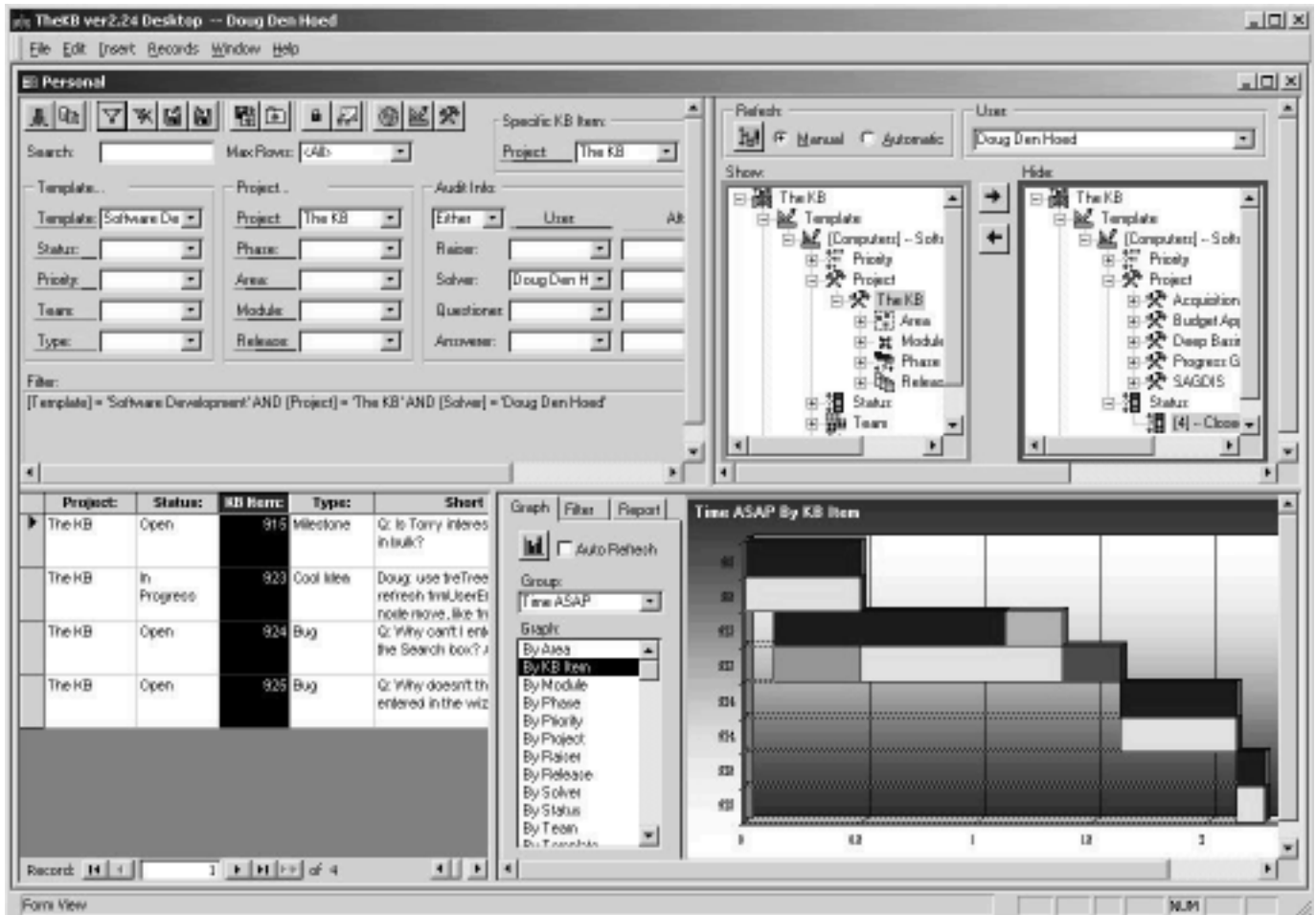


Figure 3. The KB in docking station mode.

time against the project on that form before closing the form. Double-clicking the KB Item column will sort the tasks from oldest to newest.

In the bottom right subform, the user selects the Time ASAP group to indicate the kind of chart that he or she wants to generate. I coined the “ASAP” term to represent a class of Gantt-like charts that display a critical path of remaining work, summarizing the earliest time to finish. I use ASAP charts to answer questions from my clients such as, “Doug, if you drop everything but X, could you finish X by next Friday?” Within the ASAP group, the user chooses to display the By KB Item chart and then clicks the chart Refresh button.

The chart shows a lot of information, distinguished by color. For each KB Item, there are two sets of bars. The upper bars show the original (blue) and revised (cyan) budgets, in hours. The lower bars show the actual to date and under budget (green), the estimate to complete and under budget (yellow), and the portion over budget (red). The ASAP chart lines up only the work yet to be done to drive its critical path. Item 915 will take half a day, and then it lines up the remaining portion of item 923. Then, when 923 is finished (over budget, incidentally), items 924 and 925 are shown, projecting an overall 2.25-day duration.

This dynamic chart generation is made possible by temporary tables. I use CSV files to hold the data that drives the ASAP charts. After the users have made their selections, all of the KB Items are present in the datasheet, filtered, sorted in the right order, with their budget, estimate to complete, and actual amounts. So, The KB code just walks the datasheet’s recordset to load the CSV file with entries for each KB Item, including some extra columns to drive the bars on the ASAP chart. The ASAP chart then joins the information in the temporary tables to the real tables, grouping to roll it up when requested, and displays the chart.

Storing transient data

Another way I’ve used TempCSVs is to simulate a row-level view of data. My team developed this technique during our first replication project back in Access 95 BPR (Before Partial Replication), when we had to send all data everywhere. We used a temporary Jet table to hold data that controlled what the current user was permitted to see, and then we joined that control table to every form, report, and recordset in the application. The join restricted the data to only the rows that pertained to the current

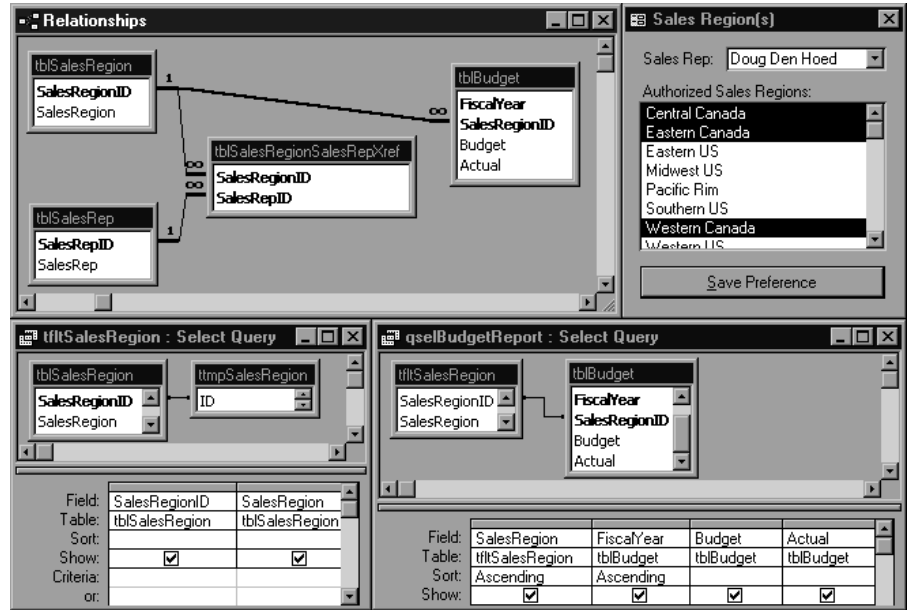


Figure 4. Horizontal segmentation of data.

user, an effect we called “horizontal segmentation.”

In Figure 4, you can see an extension of our original concept. In the top left corner, the data model shows the many-to-many relationship between Sales Regions and

Continues on page 21

If I save the file using a .txt extension instead of .Act, this works fine. However, saving to a file with .Act produces error 3027 stating that the database is locked. I found that most extensions have been disabled. Is there a good reason? They should at least include a better error message! It was painful to locate the solution.

Unfortunately, in an effort to close up security holes, Microsoft sometimes makes changes that break existing code (ask anyone who's written code that automates Outlook!). The purpose here was to keep folks with more brains than sense from being able to create diabolical Access viruses that would destroy important system files. But in this case, the new security feature is completely within your control, once you know where to look in the Registry. A new Registry key was added that allows you to override the new limitations on which text files can be manipulated by Jet. You can add .Act back in by editing the following Registry key:

HKEY_LOCAL_MACHINE\Software\Microsoft\Jet\4.0\Engines\Text

Change the value named "DisabledExtensions" to:

!txt, csv, tab, asc, tmp, htm, html, act

Note that this list begins with an exclamation point, which is used here as a logical Not operator, so even though the Registry value is called "DisabledExtensions," the list actually includes the enabled ones. ▲

Contributing Editor Andy Baron, based in Singer Island, FL, is an independent developer, a senior consultant with MCW Technologies (www.mcwtech.com), and a trainer for Application Developer's Training Company (www.appdev.com). Since 1995, Andy has been selected by Microsoft as an Access MVP, based on his contributions to Microsoft's online support. Along with Mary Chipman, Andy is co-author of *Microsoft Access Developer's Guide to SQL Server*. Andy_Baron@msn.com.

Temporary Tables...

Continued from page 9

Sales Reps. The top right corner shows a preference form (frmSalesRegionPreference in the sample database), where a user can select the Sales Region(s) to work with. The Save Preferences button calls tmpCreateFromListbox, killing, creating, loading, and linking a CSV table called ttmpSalesRegion that lists the selected Sales Regions. The bottom left corner shows the filtering query (tfltSalesRegion), where the ttmpSalesRegion table restricts the master tblSalesRegion table to the user's current preference. This query, tfltSalesRegion, can be used in a form or report, or in place of any recordset, that's "preference-specific." The bottom right corner shows an example of a query (qselBudgetReport) that might drive a preference-specific budget report.

Although we now use partial replication on the project I mentioned, we've used temporary tables to store transient data and provide horizontal segmentation of data on many projects since then.

Solving concurrency issues

My last example demonstrates how I've used CSV files to allow many users to simultaneously run the same reports with varying criteria.

For performance reasons, I always install the MDB that contains my application's front end (forms, reports, and so forth) on each user's desktop. This is faster than pulling forms and reports over the network. My front ends generally have a single Report Submission form like the one shown in [Figure 5](#) (on page 22). The top left combo

box is bound to a table that lists available reports. When you select a report, the combo box's AfterUpdate event fires, checks the combo box's hidden columns, and enables the list boxes as appropriate for the selected

report. You can then select the criteria for the report from the list box. When you click Preview or Print, the code creates CSV tables for each list box, including the disabled ones (this is why the code in my function treats disabled list boxes as if all of the items in them were selected).

Most of what look like different reports to my users are actually the same underlying report. Each report's query consults a function to perform a dynamic Group By, depending on which version of the report has been selected. I link every CSV that might need to be restricted, knowing that if that list box isn't enabled, every value will be present.

It sounds complicated, and it sounds like a lot of work. Okay, it was. However, it also means that I have only five real reports to maintain, but my users see 90 versions. The reports run fast, too: The longest running report completes in less than 10 seconds with realistic volumes, regardless of what Group By has been chosen. And, because the application front end is local, the users each get their own CSV tables and can run the same reports without affecting other users' criteria.

These examples should give you some ideas to extend query functionality, store transient data, and solve concurrency issues in your applications. I invite you to use my tmpCreateFromListbox function and to

Figure 5. A consolidated report submission form.

experiment with my TempCSV technique. Finally, pay close attention to the way your code causes database bloat. And keep compacting! ▲

DOWNLOAD [CSVTEMP.ZIP at www.smartaccessnewsletter.com](http://www.smartaccessnewsletter.com)

Doug Den Hoed is a founder of Lumina Systems Delivery in Calgary, Canada, which specializes in customized software solutions using Access, Visual Basic, InterDev, SQL Server, and Oracle. Doug recently released The KB™ (www.thekb.com), the commercial Access application that inspired this article. doug.denhoed@home.com.

Downloads January 2001 Source Code

- [CSVTEMP.ZIP](#)—Doug Den Hoed provides a Christmas gift with a sample application that uses comma-delimited files to store temporary information. The application includes a standard routine for loading CSV files with data from a list box. Finally, the package includes a sample application that demonstrates how typical Access activities can cause your MDB to expand. (Access 97)
- [TBLMENU.ZIP](#)—Keith Bombard has sent along all of the administrative forms for his user- and table-driven menuing system. The sample application shows how menus are generated from the tables in the system. (Access 97)
- [LCEVENTS.ZIP](#)—Peter Vogel's download shows how to use Loosely Coupled Events to break your Access application into two parts that can process asynchronously. You'll need Visual Basic to compile some of the code. (Access 97)
- [SA0101.ZIP](#) and [SA0101.CHM](#)—The latest update to the cumulative *Smart Access* index comes in Access 2.0, Access 95, and Access 97 versions. It includes full descriptions of every article since the January 1999 issue of *Smart Access*, as well as an index to each month's Developer Disks/Subscriber Downloads. (The index is available in "SA0101.ZIP—The Complete January 2001 Source Code.")