PhUSE 2011

Paper DH01

Don't Bring a Spreadsheet to a Database Fight: A Case for Microsoft Access

Ryan Burns, Rho, Inc., Chapel Hill, USA

ABSTRACT

Microsoft Excel is a popular spreadsheet program used by professionals in the pharmaceutical industry from programmers to project managers. In most cases, there is little to no startup time when creating a spreadsheet and the results can be very pretty. These characteristics make Excel a product that is hard to pass up. However, it is often misused as a database platform. Luckily, Microsoft also makes a database program: Access. Access allows for more structure and easier sharing. It is also far superior if the data will be read or written by SAS®. This paper will make a case for using Access instead of Excel and will also present some examples for setting up and using Access databases.

INTRODUCTION

Pharmaceutical programmers work with and create many types of data. This data may be related to clinical data, such as lookup tables used in data mapping, or it may be related to business tasks, such as project tracking. When dealing with clinical data, SAS is the default tool for most programmers. For non-clinical data most people look first to Microsoft Excel because they have a copy of it, they have experience using it, and they can produce something that looks nice relatively easily. So, why not use it? For one, it is a spreadsheet—not a database application. Fortunately, most installations of Microsoft Office now include Access: a flexible, robust, and user-friendly database application. Compared to Excel, Access requires users to spend more time at startup, is less omnipresent, and produces less pretty results. However, in most cases these drawbacks are far outweighed by increased structure, easier sharing, and improved interactions with SAS.

LOOKUP TABLES

Recoding and categorizing data are common programming tasks. In many cases, these are trivial tasks that involve only a handful of original values. However, there are datasets that start with scores of values that need to be handled. The most common example of this is lab data. When dealing with a large number of tests and test codes that need to be standardized and grouped, manually keying each recode is tedious and error prone, so external lookup tables are often used.

These lookup tables are created to be functional more than presentable and are often put together rapidly as they are needed. One way to create them is to create a simple Excel worksheet with a few needed columns, such as "old test name", "new test name", and "category". Programmers can then import the spreadsheet into SAS. Then they just have to hope that no one has the spreadsheet open while they run their program, that SAS properly guesses column attributes, and that the program will be able to separate the data rows from the header rows. It may be simple to create a lookup table in Excel, but there is more to consider than just ease of setup.

Another way to create lookup tables is to use Access. A simple, one-table database can take the place of an Excel worksheet. Using the Jet database engines, SAS can interact with any Access table or query. When importing from Access, SAS will run successfully if the database is open as the program runs, attributes are also transferred, and the data and descriptor portions of the table will already be separate. In addition, Access offers a view that is similar to that of Excel and it requires only slightly more effort at setup. The example that follows will show how to create such a database and illustrate that in addition to being powerful and useful there is no reason to be afraid of Access.

LAB TEST LOOKUP TABLE EXAMPLE

- Open Access and create a new blank database—unlike other Office application, you must specify a file name and location before the database is created
- A database will open with a blank table (Table1) displayed—the table can be renamed; this paper will refer to this table as TabNames
- Switch the view to Design View—it is possible to start entering data in the table without changing the view, but explicitly creating columns/fields in Design View is preferable
- Design View is used to create fields and set their attributes
    - By default there will be a variable ID which is the primary key and has type Auto Number; this can be renamed, but should be kept

  - The data type should be thoughtfully set for each field
  - In the lower section other attributes can be set—the most important are Field Size, Required, and Indexed (especially if duplicates should not be allowed)
- For TabNames, in addition to ID, the following fields should exist

Field Name	Data Type	Field Size	Required
oldName	Text	255	Yes
newName	Text	200	No
newCode	Text	8	No

- Switch the view back to Datasheet View and enter data
  - Data can be pasted in or typed
  - The ID field will populate automatically for each row
  - Adjust and hide columns to suit data and personal preference

The above process requires only a couple of minutes to complete, but can save hours of headaches down the road. Additionally, some time can be saved because there is no need to enable sharing or even save changes as data is entered. Access requires changes to be saved to database design—via manual request or option button when prompted—but changes to data are saved automatically whenever the focus moves from the edited row or a table or database is closed.

QUERIES

In the previous example, SAS would read the data table directly. SAS is also capable of reading Access queries. Queries in Access are similar to data views in SAS and usually represent a subset of a table or the result of joining multiple tables. Access provides multiple methods for creating queries: a point-and-click interface, a wizard, and SQL. Programmers with knowledge of SQL can use that method to create queries as they would anywhere, and people without knowledge of SQL can still create queries using one of the other methods—do not worry, the dancing paper clip should not appear.

Continuing the example of working with lab data, we will now work on adding categories to the data (Hematology, Urinalysis, &c.). Assume there is a table that already exists, TabCats, which contains a list of lab test codes and corresponding categories and is used with multiple projects and lab vendors. It is possible to add a field to TabNames and populate it with the information from TabCats. Populating that field would be error-prone and redundant. A better option would be to create a query that joins TabNames and TabCats. The following SQL code could be used to create the query:

```
SELECT n.oldName, n.newName, n.newCode, c.labCat
FROM TabNames as n left join TabCats as c
  ON n.newCode = c.labCode;
```

If a user viewed the resulting query in Access or read it with SAS it would appear the same as a table with fields for old name, new name, new code, and category. There are other ways to accomplish the same result, such as merging datasets in SAS, but it is not advisable to attempt to use Excel for this task.

Recoding and categorizing lab tests are relatively simple tasks. However, it is still a data-driven task, so a database is a more appropriate tool than a spreadsheet. The added costs for using Access instead of Excel are minimal, if they exist at all, and the benefits are clear.

APPLICATIONS FOR NON-CLINICAL DATA

When non-clinical data exists, more times than not Excel, with all its colors and variable cell sizes, is used. Allowing people to enter data anywhere on a spreadsheet, using any format, and then trying to analyze that data, can be a nightmare. Many people would rather compile hand-written notes and make all calculations using a slide rule. Alternatively, Access provides a platform that can keep everyone smiling. Programmers can create an Access database that is restrictive where it needs to be, flexible in other places, and user-friendly. Data can be collected in several ways including a data entry screen, a tabular form, or imported from some other data source. At setup, reports and queries can also be created and viewed by programmers and non-programmers once data has been entered. Finding a balance between the technical requirements of a database that concern programmers and the aesthetics and usability of a database for non-programmers can be tricky. Excel leans too far towards the latter; employing only SAS probably leans too far towards the former; and Access, like mama bear's porridge, is just right.

This paper has touched on tables and queries in Access. Another core component of Access databases is the form. Forms provide a way to control data entry, the ability to restrict viewing and changing the database, and a visually pleasing user interface. Forms can be used for different purposes such as hiding some of the guts of a database from users that do not need to see them or integrating queries, tables, and reports into one interface. The complexity of forms can also vary widely from a simple form that looks like a standard tabular datasheet to complex forms with subforms, controls, and macros. It is easy to get carried away when creating forms, but when done appropriately and in conjunction with good database design, forms help make Access a powerful and useful tool for developers and end-users alike.

One place where a non-clinical database can be useful is the tracking of projects and budgets. This paper will walk through the creation of a database that tracks current contracts including the project lead, contract cap, budget analysis, and sponsor contacts. The database to be created was originally stored in an Excel spreadsheet with no associated reports. In the end the database will be used to enter current project information, store snapshots of the data, and produce several reports without the use of any external applications.

PROJECT TRACKING DATABASE EXAMPLE

Task requirements:

- The following fields need to be present: Project Code, Sponsor, Project Lead, Sponsor Contact, Budget Cap, Cap Type (money vs. time), Money Spent, Hours Spent
- Data entry should be easy, have few restrictions, and be consistent
- Must be able to store and view snapshots of the data
- Allow users to generate the following reports as needed:
    - List of project leads with active projects
    - Number of projects by project lead
    - Projects within 90% of their cap
    - All fields considered important

The first step is to create the data tables.

- Table People will store information about any people that will be used in the database; fields include:
    - Person_ID: the primary key of type AutoNumber
    - Name: a text field
    - Internal: a field to differentiate internal employees from clients of type Yes/No
    - Client: a text field
    - Title: a text field
- Table ActivityCurrent will store the latest data for all fields included in the task requirements
    - A primary key will be included
    - Project Lead and Sponsor Contact will store values of Person_ID from the People table and will have a datatype of number and field size of long integer
    - Project Lead will be selected by users from a pull-down list that is populated with the names of people from the People table who are internal employees; the following attributes under the lookup tab should be set:
        - Row Source Type: Table/Query
        - Row Source: `SELECT person_id, name FROM people WHERE internal=-1 ORDER BY name;` (-1 is how 'yes' is stored for a yes/no field)
        - Bound Column: 1 (specifies which column from the row source statement will be stored in the Project Lead field)
        - Column Count: 2 (specifies how many fields are included in the row source statement)
        - Column Widths: 0"; 3" (hides the first column selected in the row source statement)

Access is capable of generating fancy reports, but for the purpose of this example formatting is not important. In order to get the reports needed, a set of queries will be created. Anytime the queries are opened, the desired reports will be displayed as a table. The table ActivityCurrent stores project lead ID numbers rather than names, but Access is aware of this connection and will display project lead names in the query results. This is another way that Access allows developers to design a database according to their preferences without exposing disinterested—and often easily confused—end users to its internal workings.

Queries can become very complicated when multiples tables have to be joined or there are nested queries. Luckily, for this example, the queries are simple. Below are SQL statements that will yield the desired information.

- List of project leads with active projects: `SELECT DISTINCT Lead FROM ActivityCurrent WHERE active=-1 ORDER BY lead;`
- Number of projects by project lead: `SELECT Lead, Count(1) AS [Number of Projects] FROM ActivityCurrent GROUP BY Lead;`
- Projects within 90% of their cap: `SELECT * FROM ActivityCurrent WHERE iif(capType="Dollars", MoneyTotal/Cap, HoursTotal/Cap) >= 0.90;`
- All fields considered important: simple select statement including only the important fields

The most familiar, and often most useful, form style in Access is the datasheet form. A datasheet form is tabular and looks like a table or Excel worksheet. The formatting and display options in a datasheet form are not as extensive as those in Excel. For example, all rows in the form will have the same height. On the other hand, Access forms are much more powerful than Excel worksheets. A form can contain data from a table or query, which can be modified via filters and sorting. This permits data that is stored in multiple locations in the database to be displayed together. Forms also have events. Events are the actions of the user and range from when a mouse button is pressed down to when the form is closed. Specific tasks can be assigned to run anytime a certain event happens. One of the most commonly used event/action pairs is checking values before saving them. The Before Update event happens between the time a user indicates a cell or row should be updated and when it is actually updated. At that moment values can be checked and based on the results, changes can be cancelled, users can be prompted to confirm the changes, or other values can be changed as a result of the update. The possibilities for what one can accomplish using events and associated actions are almost limitless.

For this example of a project tracking database, a datasheet form based on the ActivityCurrent table will be created and called FormActivity. One way to do this: with the table open, click Create and then select Datasheet Form. This provides a datasheet form with all fields and is an excellent starting point. Since data entry should not be restricted in this case, all that is left to do is move columns around to be in a logical order and adjust their widths (or hide them) as needed. At this point the form will still look much like a table. The next step is to put this form in a container that provides the ability to carry out the other requirements of the project.

One other type of form is an unbound form, which is a form that is not tied to any data. This example will use an unbound form as a container for FormActivity and buttons that will open the various queries. The first step is to create a blank form; it will be called FormContainer. Make sure that FormContainer is displayed with Design View before continuing. It is possible to embellish the blank form with pictures or background colors, but that is not a requirement of this project. The largest feature that will be in this form is FormActivity, so that will be dragged in first and become a subform. Next, a button will be added above the subform for each of the four queries. To create a button, select Button from the Controls section of the Design menu and then draw a rectangle—if a wizard pops up, click cancel. Once one button is created, it may be easier to paste several copies of that button instead of creating new buttons from scratch. Since the wizard was not used, these buttons will not perform a function until one is assigned.

Event actions, captions, control names, and many other things associated with objects in a form can be set in the Property Sheet. The Property Sheet can be opened directly or by right clicking on any object and selecting properties. For each of the buttons there are three properties that need to be set. One is the caption (Format tab)—this is the text that will be displayed on the button. Next is the name (Other tab)—this is the name that Access will use internally to identify the button; it makes work much easier to have a name with meaning rather than a meaningless number. The last crucial property is the On Click (Event tab)—this will assign the action that will start when the button is clicked. To assign an action, click in the cell next to On Click and then click the ellipsis. Options for using different builders will be presented. The Macro Builder or Code Builder will work. The Macro Builder is a menu-based method of using internal Access macros, while the Code Builder brings up a Visual Basic editor. The VBA statement that is needed is `<DoCmd.OpenQuery "HasActive">` (replace "HasActive" with the name of the respective query for each button).

In addition to the buttons that open queries, a button will need to be added to archive the current week's data and later a pull down will be added to select the week to be displayed. Create a new button and in the properties change the caption to "Archive Week", change the name to "cmdArchive", and enter the code builder for the On Click event. The below code should be added to the newly created cmdArchive_Click subroutine:

```
   Dim weekNum As String

'Check that data displayed is for the current week.
       If Me.FormActivity.Form.RecordSource <> "ActivityCurrent" Then
           MsgBox "Archiving is only for current week", vbInformation, "Not Current Week"
           Exit Sub
       End If

   numEntry:
'Prompt user for date range of the week being archived
       weekNum = InputBox("Enter date range for week", "Archive Week")
'Check that a data range was entered, and if not, give the user the option to cancel the
  archive.
       If weekNum & "" = "" Then
           If MsgBox("Would you like to cancel archive", vbYesNo, "Cancel?") = vbYes Then
               Exit Sub
           End If
       End If

'Check that the date range entered does not already exist
       If TableExists("Activity" & weekNum) = True Then
           If MsgBox("The week already exists." & vbCrLf & "Would you like to enter a
   different week?", vbYesNo, "Duplicate Week") = vbNo Then
               MsgBox "Archive cancelled", vbInformation, "Oops"
```

```
            Exit Sub
        Else
            GoTo numEntry
        End If
    Else
'Copy current table to archive table
        DoCmd.CopyObject , "Activity" & weekNum, acTable, "ActivityCurrent"
        Me.cmbWeekSelect.Requery
    End If
```

Each time a user clicks this button and successfully enters all information a new table will be created as a copy of ActivityCurrent and stored in the database with a name like Activity[Date range].

The final requirement of the project is to be able to view the archived weeks. This will be accomplished by adding a pull down that lists all available weeks and uses that selection to change the data source of the FormActivity subform. Adding a pull down is very similar to adding a button—the difference is that instead of selecting Button from the Controls section of the Design menu, Combo Box is selected. Once the combo box (pull down) is added to FormContainer, the properties need to be set. As before, it is advisable to name the new control—in this case the pull down will be named cmbWeekSelect. The Row Source property (Data tab) should be set to:

```
SELECT sortvar, sName, name
FROM (SELECT 0 as sortvar, MAX("Current") AS sName, max("ActivityCurrent") as name FROM
msysobjects)
UNION ALL (SELECT 1 as sortvar, mid(name,9) as sname, name FROM msysobjects where type=1
and name like "activity*" and name<>"ActivityCurrent")
ORDER BY sortvar;
```

The Bound Column property (Data tab) should be set to 3. The Column Count and Column Widths properties (Format tab) should be set to 3 and <0";3";0"> respectively. The above steps will populate the pull down with a list of all tables in the database that start with "Activity" and will display either "Current" or the date range, with "Current" always first. The final step is to change the On Change property (Event tab) by going to the Code Builder and entering `<Me.FormActivity.Form.RecordSource = cmbWeekSelect>` and `<Me.FormActivity.Requery>`, on separate lines. This will update and refresh the FormActivity subform each time the selected value of the pull down changes.

Everything can now be saved and the required tasks will be completed. There is one more step that makes life easier for end users—setting the database to open FormContainer on startup. Under Access Options there is a section for Current Database. Within that section is an option Display Form with a pull down. Select FormContainer from the list and that form will open anytime the database is opened. If it is necessary for development purposes to open the database without opening the form, simply hold down Shift while Access is starting. This project tracking database has some complicated components, but end users will only see a single form that seamlessly does everything they need. Furthermore, while the data entered could be garbage, at least everyone's name will only be spelled one way and budget numbers will only be entered as numbers.

CONCLUSION

Access is a tool and therefore not a replacement for common sense. However, it is a flexible and powerful tool that ought to be considered. As has been shown, an Access database can be simple and straightforward or it can be a more complex entity with many parts and a user-friendly shell. Excel is not a bad or evil product—it simply is not made to house a database. The next time you open a new Excel workbook, whether it be to recode lab tests, track contracts, or plan a party, I hope that you will ask yourself, “Do I want to create a database to store and perhaps analyze this data or do I want to present something that’s in a nice grid?” If you decide that a database is desired, consider closing Excel and opening Access in its place.

ACKNOWLEDGMENTS

Een speciaal woord van dank aan Liedeke Allyn Sharp, zonder wie dit papier zou niet half zo leesbaar. I would also like to thank Jeff Abolafia and Susan Boyer for their encouragement and support.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Ryan Burns
Rho, Inc.
6330 Quadrangle Dr
Chapel Hill, NC 27517
Work Phone: +1 (919) 408-8000
Fax: +1 (919) 408-0999
Email: rburns@rhoworld.com
Web: www.rhoworld.com