Upgrading your Access Application for a Multi-user Environment

by
Peter Lavin

June 20, 2004

Overview

MS Access is not an industrial-strength database like Oracle or DB2 and nor does it pretend to be. It is best suited for desktop applications. However, it is often the case that an existing application needs to be upgraded for a multi-user environment. Depending upon circumstances, it is not always necessary to implement a true client/server RDBMS. If demands are fairly modest Access can function satisfactorily in a multi-user environment.

This article will deal with the issues that arise when upgrading an Access-based application.

Introduction

It often happens that a database designed for a single-user needs to be upgraded for use in a multi-user environment. This article will show how to upgrade your single-user, Access-based application. As a rough rule of thumb you can continue to use Access if your total number of users does not exceed twenty. Any more than this and you should consider changing to a more robust client/server database.

The three major issues that arise when converting an application for a multi-user environment are related to concurrency, security and performance. This is what we intend to address in this article. It is assumed that you are using MS Access 2000 or higher.

First a note about concurrency. The discussion of concurrency assumes you are using a Visual Basic (VB) front-end with ADO data objects. If this is not the case then the concepts discussed in this section may apply but not the details. Likewise, concurrency problems only arise when multiple users can edit and update information in a database. If your desktop application simply allows users read-only access to data, and for example, updates are done after hours in exclusive mode then, again, this section will not concern you and may safely be skipped.

Concurrency Issues

Concurrency issues arise in a multi-user environment because two users may want to simultaneously update the same record or delete what another wants to edit. This is a common programming problem that is overcome through the use of a monitor or a lock.

We may also want users to see changes in real time. In this case the type and location of the cursor are important. We will begin our discussion by describing the kind of connection object and recordset object that are needed.

Connection Object

Let's start by discussing the connection object and what properties it should have. For a discussion of concurrency the most important feature is the "Mode" in which a database is opened. While Access defaults to opening a database in **adModeShareDenyNone** – a non-exclusive mode that will allow the user to both read from and write to a database it does not hurt to make this explicit when opening a connection. (Of course you need not open the database in this mode if it doesn't make sense for the current user). Find below the code for opening a connection.

```
Dim strpassword as String
Dim struser as String
Dim strConnect as String
Dim cnDemo as ADODB.Connection

'assume code to retrieve username and password
'from a dialog box

Set cnDemo = New ADODB.Connection
strConnect = "Data Source=" & App.Path & "\dbname.mdb;User ID=" & _
  struser & ";Password=" & strpassword & ";Provider=MSDASQL.1"
With cnDemo
    .Mode = adModeShareDenyNone
    .ConnectionString = strConnect
    .Properties("JetOLEDB:SystemDatabase")=App.Path & "\Secured.mdw"
    .CursorLocation = adUseServer
    .Open
End With
```

You cannot open a recordset that allows multiple users to change data in the database unless it is opened against a connection that is shared and read/write. Cursor location is also important when opening your connection. When using a server-side cursor changes made by other users are visible. Since the ConnectionString and Properties relate to security, they will be discussed later.

To summarise, the connection type is most important in the context of concurrency because no issues will arise if the mode is exclusive or if users are not allowed to change data in the database. Additionally, if a recordset is opened against a connection that is opened in the **adModeRead** mode then no matter how you open your recordset you will not be able to change data and hence issues of concurrency will not arise.

Recordset Object

If you want the user to navigate through your data, using a recordset object is the commonest way of doing this. Now, a recordset doesn't need to be opened against a connection object as such. It can be opened by passing in a string parameter that describes a connection. However, it is better programming practice to use a connection object because the programmer has an explicit reference to the object and can set it to "nothing" if need be. For this reason we recommend opening recordsets against existing, explicitly created connection objects.

Find below code to open a recordset against the connection object created above:

```
Set rsBooks = New ADODB.Recordset
With rsBooks
   .ActiveConnection = cnDemo
   .Source = "Select * From tblItems"
   .CursorType = adOpenDynamic
   .LockType = adLockOptimistic
   .CursorLocation = adUseServer
   .Open
End With
```

In order to see changes made by other users we must use a dynamic, server-side cursor. As an aside, CacheSize also has an effect on visibility of changes but only if its default value of “one” is changed.

The default lock type is read-only so we must explicitly use some kind of record locking if we wish to update or change records. We have a choice of using optimistic or pessimistic locking. The difference between these two types, as described by Microsoft, is that a pessimistic lock, “Prevents other users from accessing data while locked”.

Please note that the CursorType is dependent on the CursorLocation. If the cursor location is set to the client then the cursor is static regardless of how it is defined in the CursorType.

Security

Moving from a single-user to a multi-user environment makes plain the need for security. The single most important thing that you can do to secure your database is to use the security features of Access and create users and groups with various rights and permissions.

This is done in Access by creating a new workgroup, a database sometimes referred to as a system database, to describe the rights and permissions you wish to ascribe to various users and groups. Workgroup files are identified by the extension ”mdw”. The default workgroup file is usually located in the “Program Files\Microsoft Office\Office” directory. In most cases it is not desirable to change this file but rather to associate a specific database with a newly created workgroup. The simplest way to do this is to create your workgroup using the “User Level Security Wizard” under the “Tools” and “Security” menu options. Before proceeding I would strongly recommend that you first create a backup copy of your database.

Workgroup files are handled differently depending upon whether you are using a VB front-end to your database or a front-end created in Access. If you are using

VB then you will need to set the System database property. Let's repeat the code shown earlier:

```
Dim strpassword as String
Dim struser as String
Dim strConnect as String
Dim cnDemo as ADODB.Connection

'assume code to retrieve username and password
'from a dialog box

Set cnDemo = New ADODB.Connection
strConnect = "Data Source=" & App.Path & "\dbname.mdb;User ID=" & _
  struser & ";Password=" & strpassword & ";Provider=MSDASQL.1"
With cnDemo
    .Mode = adModeShareDenyNone
    .ConnectionString = strConnect
    .Properties("JetOLEDB:SystemDatabase")=App.Path & "\Secured.mdw"
    .CursorLocation = adUseServer
    .Open
End With
```

As you can see the system database property is set to a workgroup database called "Secured.mdw" located in the same directory as the VB application. In order to open the database this workgroup file must be identified and the user and password in the connection string must be found in this workgroup file.

If you have created your application using Access only, then you will need to take a different approach to using a workgroup file. To open a database that uses a workgroup file other than the default, requires setting up a short-cut to your database application using the command line switch "/wrkgrp" and specifying the ".mdw" file you wish to use with this application. You cannot simply double-click the database and open it. When setting up a workgroup using the wizard this shortcut will be created for you but if you wish to do it manually make sure your shortcut follows the instructions above and specifies the path to "Access" with the full path to the database as a parameter and *not* the path to the database itself.

While there is not space here, nor perhaps the necessity, for an in-depth discussion of how user and group rights are assigned some further comments are required. Using the wizard will force creation of a database owner with the name of the current Windows user. This owner will have administrative rights over the database. The wizard will also remove "Admin" from the "Admins" group and suggest that the "Users" group have no permissions. These are all very necessary steps in securing your database and it's certainly worthwhile researching them further if you do not fully understand the implications.

Performance Improvements

One of Microsoft's recommendations for improved performance across a network is database-splitting. With database splitting static data is left on the client's machine and changeable data is moved to a network drive. What this amounts to is creating a new database on a network drive and moving all tables into this database. Each client database will hold queries, forms, VBA code and reports with links to the tables on the shared network drive. This will reduce network traffic and speed up queries and reports. It will also reduce contention for any temporary objects created. This technique is useful regardless of whether you developed your application using a VB front-end or using only Access. In some instances it may also make sense to keep tables on the client rather than move them to the networked drive. For instance, if you have a look-up table of city names that rarely changes it might make sense to keep this table in each client database.

If you do decide to go this route Microsoft has made it very easy by creating a database utility to perform this function. Before proceeding though you will want to make use of another wizard – by analyzing performance under the "Tools" menu option. This tool will examine your database and come up with suggestions for improvements. Most important are the creation of additional indices. In a networked environment this will help to reduce network traffic.

So far we have concentrated on server-side dynamic cursors and the accompanying concurrency issues. However, for the sake of performance we should also use static, client-side cursors whenever appropriate – when running reports for instance.

You can also compile your database using the menu option to create a MDE file. This option really only makes sense if you are using Visual Basic for Applications code in an Access based application and has more bearing on security than performance.

Other Issues

Implementation of error trapping in a multi-user environment also requires special attention. For example, provision should be made for retrying updates that fail due to conflict over resources. However, providing a thorough review of error trapping is perhaps the subject for another article.

You will also want to encrypt your database. This may have already happened depending upon which wizards you have used to this point . If not, it is a fairly straightforward matter of using the correct menu option.

Finally, if you have created your application using VB then you will want to deploy it using the Package & Deployment Wizard that comes with Visual Studio.

Conclusion

This article has taken you through the major steps necessary to upgrade your desktop Access database application for a multi-user environment. Concurrency and security issues, were covered as were performance improvement suggestions. With the right approach, an Access database can function satisfactorily in a multi-user environment.

About the Author

Peter Lavin runs a Web Design/Development firm in Toronto, Canada. For more information visit http://www.softcoded.com. He may be reached at peterlavin@sympatico.ca.