

Re: Homegrown synchronization

Source:

<http://www.tech-archive.net/Archive/Access/microsoft.public.access.replication/2006-12/msg00161.html>

- *From:* "rdemyan" <rdemyan@xxxxxxxxxxx>
 - *Date:* 27 Dec 2006 19:46:16 -0800
-

David W. Fenton wrote:

"rdemyan" <rdemyan@xxxxxxxxxxx> wrote in
news:1167266561.984368.281860@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:

..

Here's a suggestion:

Have a separate front end with linked tables to your shared database. Have the synch program use that back end, instead of the real shared back end. Then, before it runs, have it check to see who is using that front end, and if anybody else is using it, prohibit it from running. You might also do things like try to delete the LDB file, which would mean that it's actually **not** in use. You can use the ADO UserRoster for this, or the LDBUSR.DLL (look that up on Microsoft's website, or on the Access Web, mvps.org/access).

You totally lost me on this one. By a separate front end, you don't mean my main application, do you? Are you suggesting that the sync program ("front end"?) I wrote should link to the shared backend file on the server?

Ah, yes, just thought of a different method: store the synch app on the server, and use **its** LDB file to figure out if it's already in use. That makes more sense. And you could make sure it launches exclusively.

Store the sync app on the server.... hmmm..... So the idea is that if it is on the server, I could set the option to have it open

Re: Homegrown synchronization

exclusively. Is your concern that if two users have the synch app open on their machines and both synch apps try to process an update that they find, that that will cause a problem. Frankly, I hadn't considered that!

I foresee an issue with this -- you'll never know for certain when you can delete the files in the export dropbox.

Yes, clean up of these temp updateDBs will be an issue. How does Jet indirect synchronization handle this? It's sending message files, isn't it? Couldn't I just copy the technique used there?

Seems to me the import update process should create the export database. That way, each import creates a new export. This would be like a Jet replication generation number.

Yes this is my current plan. When the sync app processes an import to the shared backend, it then creates a temp updateDB based on the shared backend for EXPORT to the remote users.

Ah, I see the issue. The problem is updates to the back end by LAN users, which aren't done by imports. I'd use my synch program to run an export on a schedule, then (basically the same way you'd do with a replication hub and a production replica).

Actually updates by LAN users is an issue that occurred to me after I posted previously, and I've been wondering how to handle that. Scheduling has its advantages, but it could also mean a lot of updating of remote user apps that are unnecessary. On the other hand, if there is virtually no data in the temp updateDB sent to the remote users, then that import will only take a few seconds, which would not be noticeable since my app takes about 15 to 20 seconds to load anyway. But the file copy from the server to the remote machine will probably take longer. Originally I had thought that the model for the temp updateDB would be relatively small. It's actually 2.5 MB when it is completely empty of data. So, it's at least 2.5 times larger than I thought. Copying it to remote machines over a WAN might take a little time (but is probably not worth worrying about).

Re: Homegrown synchronization

Yes. There's be no separate process required for deletions.

But that's pretty complicated.

I'd just go with the delete flag and not worry about ever purging the actual records.

The downside, of course, is the code changes that will be required to my app.

Here's where a tool like Speed Ferret can come in handy. It allows you to do global search and replaces, so you could write a query that filters out the deletes in a table, and then search and replace for every place where the table is used and replace it with the query.

Another alternative is to rename the base linked table and replace it with a query named the same as the original table link that filters out the deletes. This would mean you don't have to edit everything.

Yes, I thought of this too because I already have RWOP queries. At first, I thought I could just modify the RWOP queries, but this won't work because the users can't get at the tables directly with the security. The only way they can get at the DeleteFlag is through the RWOP queries. So, I'd have to rename the existing RWOP queries with a prefix and then create new queries with the old names of the RWOP queries and adding "WHERE DeleteFlag = False). This will add another layer of queries (pushing it up to about 300 total in the main app).

If I implement the delete flag system, I will do this though, as there is no way I can go through all of my code to find all of the select statements, etc and modify them. Speed Ferret could help with this as well??

Further I have to add a field to each table (actually I know how to do this in code, so it's not really a big deal). But not only does the delete code in the app have to be modified, but all select code does as well. If it had been designed in at the beginning, not much of a problem. But at this stage, this seems like a tremendous amount of work.

Not with Speed Ferret. It costs about \$100 and will pay for itself the first time you use it. It's made by Black Moshannon Systems.

Re: Homegrown synchronization

I'm going to buy this anyway. I've heard good things about it.

.