

Re: Recordset.AddNew and the recordset object's data retaining

Source: <http://www.tech-archive.net/Archive/Data/microsoft.public.data.ado/2004-12/0096.html>

From: Jim Rodgers (*noway_at_jose.com*)

Date: 12/09/04

Date: Thu, 9 Dec 2004 05:32:17 -0500

You may be right about the transaction. I noticed a 10x improvement in speed when I added the transactions to an operation like yours (where I loop around data from Excel to put it into a table elsewhere). I merely assumed the improvement was due to local caching of the data, thereby avoiding the round trip per each loop. Now that you mention it, a batch cursor would work that way, wouldn't it?

Perhaps what you are saying is that you don't want to process all 10,000 (or 100,000) records at once. Then isn't the problem with how you page your source rather than how you write to the destination table? Of course, both could be problematic if you don't have the physical memory to accommodate the TWO sets of 100,000 records: source AND destination!

Anyway, regarding the suggested technique involving .Requery, I think you may have a good idea there. Furthermore, if you can use a batch cursor (or transactions or whatever) to hold down the round trip overhead between queries, then you are doing a good job. Finally, if you can "page" your way through the source data – perhaps using some kind of fancy asynchronous technique – then you might be pipelining—in data in the background while you are batching it out in a foreground process. This is the kind of effort I would pay TWO engineers to work on for a high volume shrink-wrap product (or a high-dollar consulting project at a Fortune 500 client).

Otherwise, you should do some benchmarking to assess the need for implementing a high performance solution on this assignment. I get pretty good results from 1GB RAM on a 2.5GHz P4 and a 100BaseTx LAN. Good hardware is cheaper than people's time in many cases. I hate bustin' my ass for some cheap SOB who won't replace his 400MHz P3 with 128MB RAM and a 100MHz FSB. It just doesn't make any sense.

Now, if you have 100,000 records with 15 fields at an average of 48 bytes each, that's still only 72MB. (The video card might use more RAM.) This suggests you should do some testing before you worry too much. On a workgroup application, I wouldn't think twice about filling a recordset with "SELECT * FROM InvoiceLines;" – which pulls up 150,000 records.

microsoft.public.data.ado: Re: Recordset.AddNew and the recordset object's data retaining

It seems to me your real liability is the round trip database time. Yanking 10,000 records at once is no big deal, but looping through 10,000 records with one database update trip PER RECORD is catastrophic. FIRST, make sure you don't eat that one! Then test it. Next, add-in your Requery trick. Then, do more tests. After all that, you might spend more time addressing paging issues to keep the memory resources under control if you need better performance.

```
'-----  
Set rsMyAdodbRecordset = blah, blah  
'-----  
' ? cnMyAdodbConnection.BeginTrans  
'-----  
' NO DATABASE UPDATE TRIPS during this Loop !  
Until MyBigFatLoop = IsDone  
  rsMyAdodbRecordset.AddNew  
  rsMyAdodbRecordset("Field1") = blah  
  rsMyAdodbRecordset("Field2") = blah  
  rsMyAdodbRecordset("Field2") = blah  
  rsMyAdodbRecordset.Update  
Loop  
'-----  
' ...whatever  
rsMyAdodbRecordset.UpdateBatch ' ?  
' ? cnMyAdodbConnection.CommitTrans  
'-----  
rsMyAdodbRecordset.Close  
Set rsMyAdodbRecordset = Nothing  
'-----
```

GOOD LUCK.

– Jim Rodgers

"Jiho Han" <jhan@infinityinfo.com> wrote in message
news:354046632380927385921251@msnews.microsoft.com...
> *A couple of observations to your approach...*
>
> *The statement you make about the individual updates not hitting the
database*
> *until the transaction is committed is totally different from my
understanding*
> *of how ADO works. The behavior you specify, if I'm correct, has nothing
> to do with the updates being in a transaction but rather on whether you
specify*
> *LockType to be adLockOptimisticBatch and use UpdateBatch method at the
end.*
>
> *But even if things worked as you say it does, it doesn't resolve my issue
> with the client retaining all 10,000 records in memory(I don't mean
physical*

Re: Recordset.AddNew and the recordset object's data retaining

> *memory only – in fact, if they were, it wouldn't be that bad, it's when things spill over and the memory starts paging out to disk that concerns me).*
> *Because*
> *until you set the recordset to Nothing, all records will sit on the client machine. Maybe I'm overly concerned for nothing. But what if it were 100s of thousands of records.*
>
> *I have been thinking about this for a bit last night and thought of an alternate solution that may be ok for me.*
> *The reason I wanted to avoid Recordset.AddNew, albeit its convenience/power of ease, is that in order to avoid the above issue of retaining records in memory, I'd indeed have to set the recordset to nothing, to dispose them somehow.*
>
> *Instead, I am thinking to recycle it every certain number of records.*
Obviously
> *this isn't something revolutionary, it is how batch jobs are traditionally done anyway.*
>
> *So, I would do something like this:*
>
> *' somewhat pseudocode follows*
>
> *Dim lngCount = 0*
> *Set objRS = "SELECT * FROM ACCOUNT WHERE 1 = 0" ' This lets me get the schema quickly and without fuss.*
> *Do While Not Source.EOF*
> *objRS.AddNew*
> *objRS.Fields("NAME").Value = Source("NAME")*
> ...
> *Source.MoveToNextLine*
>
> *If lngCount = 3000 Then ' assuming we recycle every 3000 records*
> *objRS.Requery*
> *End If*
> *Loop*
>
> *That requery line is the key I think. That is a Close/Dispose and Open in one line. I'd adjust the interval depending on how the app performs. I haven't tested this out yet.*
>
> *What do you think?*
>
> > *How about this:*
> >
> > '-----

microsoft.public.data.ado: Re: Recordset.AddNew and the recordset object's data retaining

```
>> Set rsMyAdodbRecordset = blah, blah, blah
>> '-----
>> cnMyAdodbConnection.BeginTrans
>> '-----
>> Until MyBigFatLoop = IsDone
>> rsMyAdodbRecordset.AddNew
>> rsMyAdodbRecordset("Field1") = blah
>> rsMyAdodbRecordset("Field2") = blah
>> rsMyAdodbRecordset("Field2") = blah
>> rsMyAdodbRecordset.Update
>> Loop
>> '-----
>> cnMyAdodbConnection.CommitTrans
>> '-----
>> rsMyAdodbRecordset.Close
>> Set rsMyAdodbRecordset = Nothing
>> '-----
>> Whaddaya think?
>>
>> I believe you will find that the records are added locally at high
>> speed because the Update won't go back to the database during the
>> transaction. When you close the recordset and set it to Nothing, all
>> memory is reclaimed. You can use transactions with most providers, but
>> not all.
>>
>> Just one thing, you might not want to open the recordset against the
>> entire table or it just might return the whole table to local memory!
>> (Wow! if it is a big table.) Can you do "SELECT * FROM MyTable WHERE
>> MyTable.MyKey = 'nevercanhavethisvalue';" ????
```

>>

>> Good luck

>>

>> Jim Rodgers

>>

>> "Jiho Han" <jihohan@gmail.com> wrote in message
>> news:1102458848.649336.188410@c13g2000cwb.googlegroups.com...

>>> I would like to use Recordset object's AddNew method to insert a new
>>> record. However, I am afraid, whenever I do so, the new row I've
>>> inserted will be retained in memory on my client machine on
>>> subsequent inserts.

>>>

>>> Imagine, inserting 10,000 records using either AddNew/Update or
>>> UpdateBatch. AddNew is very convenient and easier to code than
>>> creating a Command object and setting its Parameters collection.
>>> With the former, I don't even have to worry about optimistic
>>> concurrency since it's supported by default although in a batch
>>> scenario such as above, I'd likely turn off concurrency support.

>>>

>>> What I don't want is though, memory bloat because of the new records
>>> I've inserted. Query or Close/Open will result in additional trips
>>> to the DB which isn't the case with a Command object. Can't have the

Re: Recordset.AddNew and the recordset object's data retaining

microsoft.public.data.ado: Re: Recordset.AddNew and the recordset object's data retaining

> >> *cake and eat it too?*
> >>
> >> *Can someone clarify?*
> >> *Thanks*
>
>