

Re: Connection types and speeds

Source:

<http://www.tech-archive.net/Archive/Access/microsoft.public.access.queries/2006-08/msg01400.html>

- *From:* "Allen Browne" <AllenBrowne@xxxxxxxxxxxxxxxx>
 - *Date:* Wed, 9 Aug 2006 22:30:12 +0800
-

OpenDatabase() allows you to execute directly on the file, and to OpenRecordset of type dbOpenTable. Both of these will be the fastest approaches. The only downside is the need to look up the path (from the Connect property of the TableDef.)

It will depend how you run your query, but I would be very surprised if an Execute returned control before it completed. If it did there would be serious problems with getting the RecordsAffected, or with executing queries in sequence (such as an Append followed by a Delete, which would be a disaster if the delete began before the append completed.)

When you are working with linked tables, you may find that you get faster results if you just OpenDatabase on the back end, even if you do not use that connection. Simply holding the file open can speed up other operations that use the file. The memory used by holding the connection open is not significant, but leaving it open unnecessarily for extended periods might increase the chance of corruption of something went wrong (e.g. network broke, power cut, ...)

--
Allen Browne – Microsoft MVP. Perth, Western Australia.
Tips for Access users – <http://allenbrowne.com/tips.html>
Reply to group, rather than allenbrowne at mvps dot org.

"Dirk" <Dirk@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message
news:EFFC0899-E0B2-4BD1-9E3F-9BFE21854318@xxxxxxxxxxxxxxxx

Allen, thanks for the very concise information.

After your reply I have experimented with some other methods:

- 1) Assigning db = DBEngine(0)(0) and using that to run a query against. This was slightly faster than db = CurrentDb, however still a LOT slower than db = DBEngine(0).OpenDatabase([path_to_db]) (...do stuff...) db.Close
- 2) When obtaining db = DBEngine(0)(0) I am unable to use a dbOpenTable recordset due to the way it apparently handles linked tables. When I use

Re: Connection types and speeds

the

`db = DBEngine(0).OpenTable([path_to_db])` method I can.

3) All recordset inserts seem to consequently have 0ms execution time, regardless of the amount of records. I suspect that these recordsets are actually written to the database while the code has regained control (asynchronously). Is that true? If so, the next method to test should be negatively influenced by a preceding insert test using a recordset.

For now I am suspecting that it probably is fastest to manually open a connection to your backend database manually and use this connection throughout your session. No idea if this is efficient memory-wise though. What would you advise?

Regards,

Dirk Louwers

"Allen Browne" wrote:

Dirk, `CurrentDb` is an interesting beast.

Every time you use `CurrentDb`, Access does this:

- a) It flushes all collections in the DAO hierarchy.
- b) It creates a new object of type `Database`, and assigns it to the currently open database.

(a) represents a huge performance hump. If you trawl the ancient archives of `comp.databases.msaccess`, you will find articles by michka stating that this reference can be *thousands* of times slower than `dbEngine(0)(0)`, due to the work in flushing the collections. He also pointed out that if you do this once (not inside a loop) the performance difference is academic.

(b) is equally important to understand. If you code:

```
Currentdb.Excise "MyInsertQuery"
```

```
MsgBox CurrentDb.RecordsAffected & " new record(s) added."
```

you will always be told you have zero records added. That's because the 2nd

`CurrentDb` gives you a completely new reference that has no idea what was the outcome of the first one.

`dbEngine.Workspaces(0).Databases(0)` is faster because you are not flushing

the collections first. So, if you just created a query, you may find that it

does not yet appear under `dbEngine(0)(0).QueryDefs` yet. Additionally,

Re: Connection types and speeds

there are some occasions when dbEngine(0)(0) is *not* the current database. This can happen if you had other databases open (e.g. if you used the RecordsetClone of a form) when a transaction was rolled back, or after using wizards or compacting the database. Therefore, Currentdb is certainly safer than dbengine(0)(0).

Working with a recordset of type dbOpenTable has always been the fastest approach, but it only works with local Access tables. You cannot use this type on an attached table, or a query statement. Unless your database is a quick'n'dirty job (in which case performance probably doesn't matter much), there's a good chance that it will end up being split at some point, and then any code that uses dbOpenTable will need to be modified. So, there are very limited cases where dbOpenTable is a good choice, despite its speed.

The issue with transactions is not a cut'n'dried one. A transaction may speed things up, since the whole operation occurs in RAM until the final write, and may avoid the write completely with a rollback. But if RAM is limited or the number of records is really large, the transaction may have to be written to disk along the way, and so the intermediate writes might end up taking longer. If disk space is also limited, we could really be thrashing the drive here. It was interesting that when Microsoft released Access 97, they mutilated the dbFailOnError switch for the DAO Execute method so that it no longer used a transaction and rolled it back like Access 95 did. There documented reason for doing this was performance. Did the Access development team really dump the good behavior of this switch because they found that the transaction made the operation slower? I can't answer the question for sure, but that's what they said.

--

Allen Browne – Microsoft MVP. Perth, Western Australia.
Tips for Access users – <http://allenbrowne.com/tips.html>
Reply to group, rather than allenbrowne at mvps dot org.

"Dirk" <Dirk@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message
news:1DF2C873-3FC8-48FB-9D71-8D233BD94624@xxxxxxxxxxxxxxxxxxxx

Hi,

While doing some experimenting I noticed that using CurrentDb to reach linked tables instead of separately opening a new connection

Re: Connection types and speeds

to that
database
results in much slower queries. When I say slower I mean A
LOT.

Tests, all 200 varchar(50) field inserts and an autonumber
primary key:

ADO write: 266ms
ADO write contained in transaction: 109ms
DAO write using CurrentDb: 1578ms
DAO write using CurrentDb contained in transaction: 625ms
DAO write using new connection in workspace: 78ms
DAO write using new connection in workspace, contained in
transaction:
47ms
DAO write using a Table-recordset for inserts: 16ms

This result really surprised me on two fronts:
– First of all using the already opened CurrentDb results in
SERIOUS
slowdown for some reason. Apparently opening a separate
connection is
much
faster despite the opening and closing.
– Secondly using a recordset seems to be significantly faster
than
running
a
query directly against the database, despite the expected
overhead.

Can anyone direct me to a document outlining what happens
behind the
curtains, explaining why I come up with these results? Is it
correct
that
the
absolute fastest way to do bulk inserts on a linked table is
using a
separate
connection to that backend and use a Table recordset to do
the inserts?
Or
is
there a faster way to do this?

Hope someone can shed some light on this.

Regards,

Re: Connection types and speeds

Dirk Louwers