

Re: MS SQL Server/ODBC package for Python

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2005-04/msg02838.html>

- *From:* "M.-A. Lemburg" <mal@xxxxxxxxxx>
 - *Date:* Mon, 18 Apr 2005 18:45:44 +0200
-

Peter Herndon wrote:

:) Knock away, as my info isn't scientific anyway. In my case, ASA is **not** local. The db is running on a 500MHz x 2 server with 768MB RAM, over 100BaseT connection. That same server is also running the MSSQL instance, and IIS.

Running your benchmark, I ran into a couple of interesting points. Using mx.ODBC, my times were 0.54 seconds and 6.56 seconds respectively, while using adodbapi my results are 3.55 seconds and 25.9 seconds respectively. mx.ODBC is faster with the simple query you provide.

Next I modified the benchmark to reflect my particular circumstances more accurately (?Maybe? Comments invited). I changed the query to one of the queries in regular use in my application. This query selects 26 columns from 3 joined tables, with a where clause "where f476 = ?", and I provide the missing value as a tuple in the execute statement. Note that, as I mentioned in my reply to M-A, the f476 field is not indexed, and is a long varchar. Again, the system is bought, so I have no control over the schema. ;)

The other change I made was to reduce the number of iterations from 100 to 10. Since there are 128000 records in the main table, the wait for 100 iterations was too long for my patience. Under these circumstances, mx.ODBC's numbers are 188.49 seconds and 377.56 seconds respectively, and adodbapi's times are 111.15 seconds and 223.55 seconds respectively.

Just curious: are you timing just the time it takes to complete the .execute() or do you also fetch the complete result or only part of it ?

Note that we have had reports about MS SQL Server being very slow in determining the number of rows in a result set. It's possible that ASA has the same problem.

Re: MS SQL Server/ODBC package for Python

The mxODBC 2.0 release fetches this number after every `.execute()`. If `adodbapi` avoids this (which we'll also integrate into mxODBC 2.1), then this would explain the differences you see.

Another reason could indeed be related to the `longvarchar` field: these fields are fetched in multiple chunks if the ODBC driver doesn't provide proper size information - each of these chunks will require a network access which slows down the data fetching.

Since mxODBC supports Unicode, but defaults to returning 8-bit strings, it is also possible that your `longvarchar` column is sent as Unicode and has to be converted to an 8-bit string first. Thus, allowing mxODBC to return Unicode could make a difference as well (see the docs on how this is done).

My first wall-clock impressions are obvious exaggerations of reality, for which I duly apologize to all. However, `adodbapi` did prove to be faster in my admittedly very wacky common use case. Slower to connect, but faster to run a substantial query.

Comments? Questions? Suggestions for improvement?

See some of the hints I mentioned above.

Note that it often also pay off checking the ODBC driver settings, esp. if you have a networked setup - ODBC drivers often pre-fetch result sets and changing the defaults they use for this can make a huge difference in response times.

Unfortunately, mxODBC doesn't have control over these settings and there's no standard for them, so you'll have to check the ODBC driver documentation for details on the best settings can be found and set.

Regards,

--

Marc-Andre Lemburg
eGenix.com

Re: MS SQL Server/ODBC package for Python

Professional Python Services directly from the Source (#1, Apr 18 2005)
>>> Python/Zope Consulting and Support ... <http://www.eugenix.com/>
>>> mxODBC.Zope.Database.Adapter ... <http://zope.eugenix.com/>
>>> mxODBC, mxDateTime, mxTextTools ... <http://python.eugenix.com/>

::: Try mxODBC.Zope.DA for Windows, Linux, Solaris, FreeBSD for free ! :::
.