

Re: psycopg, transactions and multiple cursors

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2004-11/1237.html>

From: Steve Holden (steve_at_holdenweb.com)

Date: 11/09/04

Date: Tue, 09 Nov 2004 08:26:43 -0500

Alban Hertroys wrote:

> *Diez B. Roggisch wrote:*

>

>>> *AFAIK, you can't open a transaction w/o using a cursor; You need a query*

>>> *that says "BEGIN;".*

>>> *You can commit a connection object, however. It would be nice to be able*

>>> *to start a transaction on a connection object, but then you still could*

>>> *only have one transaction per connection... :(*

>>

>>

>>

>> *thats actually the case for all DBs I know including e.g. oracle with*

>> *jdbc –*

>> *so the abstraction layers usually use connection pooling to speed up*

>> *opening a connection thus the app doesn't suffer so much.*

>

Yes, most database connections will generate an implicit transaction the first time a change is made to the database (in the absence of autocommit).

>

> *Ok, that means I won't get away with a single connection object (unless*

> *psycopg puts a connection pool in a single connection object).*

>

>> *No idea why thats happening – on jdbc, one can set an "autocommit"*

>> *connection property that will do exactly that: enforce a commit if a*

>> *statement was successful. Maybe psycopg has that too?*

>

>

> *Yes, it does have autoCommit, and thankfully it can be turned off (which*

> *I did, of course).*

>

It certainly isn't too useful if you occasionally need to roll things back. Structural changes to the database will frequently cause an automatic commit anyway, though – you aren't modifying the database structure at all, I take it?

comp.lang.python: Re: psycopg, transactions and multiple cursors

>>> *I know PostgreSQL can do transactions on nested cursors, I have used
>>> that now and then in stored procedures (pl/pgsql).*
>
>
>> *I didn't find much on nested cursors on google, so I don't know how they
>> work – but I assume if they are part of psycopg, they somehow have to be
>> created using an existing cursor, as otherwise how should psycopg know
>> that
>> what you want is a nested and not a new cursor.*
>
>
> *Actually, nesting of cursors is something that PL/PgSQL can do. And so
> can PL/SQL in Oracle.
> It's something that's possible on a low database API level, and (to my
> understanding) the DBAPI 2.0 uses them for queries. It's one of Python's'
> advantages over eg. PHP, and one of the reasons I chose to use Python
> for this project.*

>
Nested cursors aren't nested transactions, though, right?

>> *So is there something on cursor object to get a new cursor, or at
>> least the
>> connection so you can get a cursor on the very same connection?*
>
>
> *A cursor is comparable to an iterator over a result set (where the
> cursor fetches one record from the database at a time).*

>
Be careful that you don't confuse the DB API cursors with the cursors
you get with DECLARE CURSOR in PL/SQL, for example. The two aren't
necessarily the same thing (and I've always felt that "cursor" was, for
that reason, not the best possible terminological choice for the DB API).

> *You use one for every query, and it can often be reused for the next
> query as well.*
>
> *However, if you loop through a result set (with a cursor) and you need
> to do a query based on the record that the cursor is currently
> positioned at, you can't use the same cursor again for that query, but
> need to open a new cursor. That's what I meant by "nesting cursors".*
>

Bear in mind, though, that it will often be **much** more efficient to do
a fetchall() from the cursor and iterate over that result. This
typically avoids many round-trips by fetching all the data at once,
though it's less practical if data sets become huge.

There's sometimes a middle ground to be found with repeated calls to
fetchmany().

In that way the cursor can be reused with impunity once the data has
been fetched.

comp.lang.python: Re: pycopg, transactions and multiple cursors

- > *That shouldn't matter for the state of a transaction, though...*
- >
- > *Maybe there's a difference between database level cursors and DB API 2.0*
- > *level cursors in Python?*
- >

As I mentioned above, there often is.

I think we've already agreed that the pycopg cursors aren't DB API compliant anyway, precisely because of the way they handle transactions. An API-compliant library shares transaction state across all cursors generated from the same connection, which (IMHO) gives the flexibility one needs to handle complex database interactions.

> *The number of questions is increasing...*

Well, the number of answers is, too, but it seems to me you **are** kind of wanting to have your cake and eat it. In previous threads you've suggested that the pycopg cursor behavior is what you want, but now that very behavior might (?) be biting you.

Anyway, you couldn't be talking to a better bunch of guys to try and solve this problem. c.l.py is sometimes persistent beyond all reasonable limits. Good luck!

regards
Steve

--

<http://www.holdenweb.com>
<http://pydish.holdenweb.com>
Holden Web LLC +1 800 494 3119