

Re: DB API 2.0 and transactions

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2005-06/msg01834.html>

- *From:* Magnus Lycka <lycka@xxxxxxxxxx>
 - *Date:* Mon, 13 Jun 2005 12:23:48 +0200
-

I'm CC:ing this to D'Arcy J.M. Cain. (See comp.lang.python for prequel D'Arcy.)

Christopher J. Bottaro wrote:

Check this out...

```
<code>
import pgdb
import time

print time.ctime()
db = pgdb.connect(user='test', host='localhost', database='test')
time.sleep(5)
db.cursor().execute('insert into time_test
                    (datetime)
                    values
                    (CURRENT_TIMESTAMP)')

db.commit()
curs = db.cursor()
curs.execute('select datetime from time_test order by datetime desc limit
1')
row = curs.fetchone()
print row[0]
</code>
```

```
<output>
Fri Jun 10 17:27:21 2005
'2005-06-10 17:27:21.654897-05'
</output>
```

Notice the times are exactly the same instead of 5 sec difference.

Re: DB API 2.0 and transactions

What do you make of that? Some other replies to this thread seemed to indicate that this is expected and proper behavior.

This is wrong. It should not behave like that if it is to follow the SQL standard which *I* would expect and consider proper.

I don't think the SQL standard mandates that all evaluations of CURRENT_TIMESTAMP within a transaction should be the same. It does mandate that CURRENT_TIMESTAMP is only evaluated once in each SQL statement, so "CURRENT_TIMESTAMP=CURRENT_TIMESTAMP" should always be true in a WHERE statement. I don't think it's a bug if all timestamps in a transaction are the same though. It's really a bonus if we can view all of a transaction as taking place at the same time. (A bit like Piper Halliwell's time-freezing spell in "Charmed".)

The problem is that transactions should never start until the first transaction-initiating SQL statement takes place. (In SQL-92, all standard SQL statements are transaction initiating except CONNECT, DISCONNECT, COMMIT, ROLLBACK, GET DIAGNOSTICS and most SET commands (SET DESCRIPTOR is the exception here).) Issuing BEGIN directly after CONNECT, ROLLBACK and COMMIT is in violation with the SQL standards.

A workaround for you could be to explicitly start a new transaction before the insert as PostgreSQL (but not the SQL standard) wants you to do. I suppose you can easily do that using e.g. db.rollback(). If you like, I guess you could do db.begin=db.rollback in the beginning of your code and then use db.begin().

Another option would be to investigate if any of the other PostgreSQL drivers have a more correct behaviour. The non-standard behaviour that you describe is obvious from the pgdb source. See: <http://www.pygresql.org/cvsweb.cgi/pygresql/module/pgdb.py?rev=1.27> (Comments added by me.)

```
class pgdbCnx:

    def __init__(self, cnx):
        self.__cnx = cnx
        self.__cache = pgdbTypeCache(cnx)
        try:
            src = self.__cnx.source()
            src.execute("BEGIN") # Ouch!
        except:
            raise OperationalError, "invalid connection."
```

Re: DB API 2.0 and transactions

```
...
def commit(self):
    try:
        src = self.__cnx.source()
        src.execute("COMMIT")
        src.execute("BEGIN") # Ouch!
    except:
        raise OperationalError, "can't commit."

def rollback(self):
    try:
        src = self.__cnx.source()
        src.execute("ROLLBACK")
        src.execute("BEGIN") # Ouch!
    except:
        raise OperationalError, "can't rollback."
```

....

This should be changed to something like this (untested):

```
class pgdbCnx:

    def __init__(self, cnx):
        self.__cnx = cnx
        self.__cache = pgdbTypeCache(cnx)
        self.inTxn = False #NEW
        try:
            src = self.__cnx.source() # No BEGIN here
        except:
            raise OperationalError, "invalid connection."
```

....

```
def commit(self):
    try:
        src = self.__cnx.source()
        src.execute("COMMIT")
        self.inTxn = False # Changed
    except:
```

Re: DB API 2.0 and transactions

```
raise OperationalError, "can't commit."
```

```
def rollback(self):
    try:
        src = self.__cnx.source()
        src.execute("ROLLBACK")
        self.inTxn = False # Changed
    except:
        raise OperationalError, "can't rollback."
```

....

```
def cursor(self):
    try:
        src = self.__cnx.source()
        return pgdbCursor(src, self.__cache, self) # Added self
    except:
        raise pgOperationalError, "invalid connection."
```

....

```
class pgdbCursor:
```

```
    def __init__(self, src, cache, conn): # Added conn
        self.__cache = cache
        self.__source = src
```

```
> self.__conn = conn # New
```

```
    self.description = None
    self.rowcount = -1
    self.arraysize = 1
    self.lastrowid = None
```

```
....
(execute calls executemany)
....
```

```
    def executemany(self, operation, param_seq):
        self.description = None
        self.rowcount = -1
```

```
    # first try to execute all queries
    totrows = 0
    sql = "INIT"
    try:
```

Re: DB API 2.0 and transactions

```
        for params in param_seq:
            if params != None:
                sql = _quoteparams(operation, params)
            else:
                sql = operation

> if not self.__conn.inTxn: # Added test

        self.__source.execute('BEGIN')

> self.__conn.inTxn = True

        rows = self.__source.execute(sql)
        if rows != None: # true is __source is NOT a DQL
            totrows = totrows + rows
        else:
            self.rowcount = -1
```

I guess it would be even better if the executemany method checked that it was really a transaction-initiating SQL statement, but that makes things a bit slower and more complicated, especially as I suspect that the driver permits several SQL statements separated by semicolon in execute and executemany. We really don't want to add a SQL parser to pgdb. Making all statements transaction-initiating is at least much closer to standard behaviour than to *always* start transactions start prematurely. I guess it will remove problems like the one I mentioned earlier (repeated below) in more than 99% of the cases.

This bug has implications far beyond timestamps. Imagine two transaction running with isolation level set to e.g. serializable. Transaction A updates the AMOUNT column in various rows of table X, and transaction B calculates the sum of all AMOUNTS in X.

Lets say they run over time like this, with | marking transaction start and > commit (N.B. ASCII art follows, you need a fixed font to view this):

```
....|--A-->.....|--A-->.....
.....|-B->.....|-B->..
```

This works as expected... The first B-transaction sums up AMOUNTS after the first A-transaction is done etc, but imagine what happens if transactions implicitly begin too early as with the current pgdb:

```
|-----A-->|-----A-->|-----
```

Re: DB API 2.0 and transactions

|-----B->|-----B->|-

This will cause B1 to sum up AMOUNTs before A1, and B2 will sum up AMOUNTs after A1, not after A2.

.