How do detect that transaction has already been started?

Asked 16 years ago Modified 2 years, 8 months ago Viewed 31k times Part of PHP Collective

30

I am using Zend_Db to insert some data inside a transaction. My function starts a transaction and then calls another method that also attempts to start a transaction and of course fails(I am using MySQL5). So, the question is - how do I detect that transaction has already been started? Here is a sample bit of code:

```
        try {
                Zend_Registry::get('database')->beginTransaction();

                $totals = self::calculateTotals($Cart);
                $PaymentInstrument = new PaymentInstrument;
                $PaymentInstrument->create();
                $PaymentInstrument->validate();
                $PaymentInstrument->save();

                Zend_Registry::get('database')->commit();
                return true;

        } catch(Zend_Exception $e) {
                Bootstrap::$Log->err($e->getMessage());
                Zend_Registry::get('database')->rollBack();
                return false;
        }
```

Inside PaymentInstrument::create there is another beginTransaction statement that produces the exception that says that transaction has already been started.

PHP php mysql zend-framework transactions pdo

Share
Improve this question
Follow

edited Nov 26, 2008 at 5:10

asked Nov 26, 2008 at 4:59
xelurg
4,304 ● 9 ● 37 ● 37

11 Answers

Sorted by: Highest score (default)

30

The framework has no way of knowing if you started a transaction. You can even use `$db->query('START TRANSACTION')` which the framework would not know about because it doesn't parse SQL statements you execute.

The point is that it's an application responsibility to track whether you've started a transaction or not. It's not something the framework can do.

I know some frameworks try to do it, and do cockamamie things like count how many times you've begun a transaction, only resolving it when you've done commit or rollback a matching number of times. But this is totally bogus because none of your functions can know if commit or rollback will actually do it, or if they're in another layer of nesting.

(Can you tell I've had this discussion a few times? :-)

Update 1: Propel is a PHP database access library that supports the concept of the "inner transaction" that doesn't commit when you tell it to. Beginning a transaction only increments a counter, and commit/rollback decrements the counter. Below is an excerpt from a mailing list thread where I describe a few scenarios where it fails.

Update 2: Doctrine DBAL also has this feature. They call it Transaction Nesting.

Like it or not, transactions are "global" and they do not obey object-oriented encapsulation.

Problem scenario #1

I call `commit()`, are my changes committed? If I'm running inside an "inner transaction" they are not. The code that manages the outer transaction could choose to roll back, and my changes would be discarded without my knowledge or control.

For example:

1. Model A: begin transaction
2. Model A: execute some changes
3. Model B: begin transaction (silent no-op)
4. Model B: execute some changes
5. Model B: commit (silent no-op)
6. Model A: rollback (discards both model A changes and model B changes)
7. Model B: WTF!? What happened to my changes?

Problem scenario #2

An inner transaction rolls back, it could discard legitimate changes made by an outer transaction. When control is returned to the outer code, it believes its transaction is still active and available to be committed. With your patch, they could call `commit()`, and since the transDepth is now 0, it would silently set `$transDepth` to -1 and return true, after not committing anything.

Problem scenario #3

If I call `commit()` or `rollback()` when there is no transaction active, it sets the `$transDepth` to -1. The next `beginTransaction()` increments the level to 0, which means the transaction can neither be rolled back nor committed. Subsequent calls to `commit()` will just decrement the transaction to -1 or further, and you'll never be able to commit until you do another superfluous `beginTransaction()` to increment the level again.

Basically, trying to manage transactions in application logic without allowing the database to do the bookkeeping is a doomed idea. If you have a requirement for two models to use explicit transaction control in one application request, then you must open two DB connections, one for each model. Then each model can have its own active transaction, which can be committed or rolled back independently from one another.

Share

Improve this answer

Follow

edited Apr 19, 2022 at 11:37

Your Common Sense
158k ● 42 ● 221 ● 362

answered Nov 26, 2008 at 6:53

Bill Karwin
561k ● 87 ● 698 ● 854

true enough, though it would certainly be an extremely nice feature. I wonder if something like that exists in Hibernate or any similar more mature persistence layers... – xelurg Nov 27, 2008 at 17:44

1 Propel has this, but I still believe it's a bogus design. See my edit above. – Bill Karwin Nov 27, 2008 at 20:12

returning back to this topic... JPA for instance has concept of TransactionManager would similar concept be logical to have for ZF? – xelurg Sep 10, 2009 at 4:35

Well I can't speak to JPA, but Java is not PHP and vice versa. PHP assumes that the *request* establishes a unit of work, and so the scope of a transaction in a PHP app should be the same as the scope of a request. It's very simple. There are no PHP application servers, so no "conversation" scope for a transaction. So there's no need to make Domain Model classes or DAO classes manage transactions -- just do it at the Controller level. – Bill Karwin Sep 10, 2009 at 6:43

@Pacerier, I don't know what "bdbaft" means. – Bill Karwin Apr 9, 2015 at 14:50

4

Do a try/catch: if the exception is that a transaction has already started (based on error code or the message of the string, whatever), carry on. Otherwise, throw the exception again.

Share Improve this answer Follow

answered Nov 26, 2008 at 6:24

John II

2

Store the return value of beginTransaction() in Zend_Registry, and check it later.

Share Improve this answer Follow

answered Nov 26, 2008 at 5:25

Imran
90.8k ● 26 ● 100 ● 131

2

Looking at the Zend_Db as well as the adapters (both mysqli and PDO versions) I'm not really seeing any nice way to check transaction state. There appears to be a ZF issue regarding this - fortunately with a patch slated to come out soon.

For the time being, if you'd rather not run unofficial ZF code, the mysqli documentation says you can `SELECT @@autocommit` to find out if you're currently in a transaction (err... not in autocommit mode).

Share Improve this answer Follow

answered Nov 26, 2008 at 5:33

Sean McSomething
6,507 ● 2 ● 25 ● 28

seems like that issue got lost in their tracker... :(– xelurg Nov 26, 2008 at 5:44

All ZF issues say "fix in next minor release" until they are actually fixed. I hope they had a good reason for doing that, because it's pretty misleading and causes confusion for a lot of people. – Bill Karwin Nov 26, 2008 at 18:34

1 In my testing via mysql client `SELECT @@autocommit;` still returns 1 during a transaction. – ColinM Nov 14, 2016 at 20:22

2

For innoDB you should be able to use

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_TRX WHERE TRX_MYSQL_THREAD_ID =
CONNECTION_ID();
```

Share Improve this answer Follow

answered Jul 5, 2015 at 13:47

curlyhairedgenius
922 ● 9 ● 8

1

I disagree with Bill Karwin's assessment that keeping track of transactions started is cockamamie, although I do like that word.

I have a situation where I have event handler functions that might get called by a module not written by me. My event handlers create a lot of records in the db. I definitely need to roll back if something wasn't passed correctly or is missing or something goes, well, cockamamie. I cannot know whether the outside module's code triggering the event handler is handling db transactions, because the code is written by other people. I have not found a way to query the database to see if a transaction is in progress.

So I DO keep count. I'm using CodeIgniter, which seems to do strange things if I ask it to start using nested db transactions (e.g. calling it's trans_start() method more than once). In other words, I can't just include trans_start() in my event handler, because if an outside function is also using trans_start(), rollbacks and commits don't occur correctly. There is always the possibility that I haven't yet figured out to manage those functions correctly, but I've run many tests.

All my event handlers need to know is, has a db transaction already been initiated by another module calling in? If so, it does not start another new transaction and does not honor any rollbacks or commits either. It does trust that if some outside function has initiated a db transaction then it will also be handling rollbacks/commits.

I have wrapper functions for CodeIgniter's transaction methods and these functions increment/decrement a counter.

```
function transBegin(){
    //increment our number of levels
    $this->_transBegin += 1;
    //if we are only one level deep, we can create transaction
    if($this->_transBegin ==1) {
        $this->db->trans_begin();
    }
}

function transCommit(){
    if($this->_transBegin == 1) {
        //if we are only one level deep, we can commit transaction
        $this->db->trans_commit();
    }
    //decrement our number of levels
    $this->_transBegin -= 1;

}

function transRollback(){
        if($this->_transBegin == 1) {
            //if we are only one level deep, we can roll back transaction
            $this->db->trans_rollback();
        }
        //decrement our number of levels
        $this->_transBegin -= 1;
    }
```

In my situation, this is the only way to check for an existing db transaction. And it works. I wouldn't say that "The Application is managing db transactions". That's really untrue in this situation. It is simply checking whether some other part of the application has started any db transactions, so that it can avoid creating nested db transactions.

Share Improve this answer Follow

answered Apr 1, 2017 at 4:33

Zac Imboden
771 ●8 ●12

1

This discussion is fairly old. As some have pointed out, you can do it in your application. PHP has a method since version 5 >= 5.3.3 to know if you are in the middle of a transaction. PDP::inTransaction() returns true or false. Link http://php.net/manual/en/pdo.intransaction.php

Share Improve this answer Follow

answered Jun 8, 2017 at 10:07

Stephen Adelakun
804 ●2 ●10 ●24

Maybe the DB layer of the framework doesn't encapsulate this PDO method for checking transaction status. I came with the same answer as you, I almost posted mine before I saw your answer here. – Leandro Jacques Sep 5, 2017 at 18:00

1

You can also write your code as per following:

```
try {
    Zend_Registry::get('database')->beginTransaction();
}
catch (Exception $e) { }

try {
    $totals = self::calculateTotals($Cart);

    $PaymentInstrument = new PaymentInstrument;
    $PaymentInstrument->create();
    $PaymentInstrument->validate();
    $PaymentInstrument->save();

    Zend_Registry::get('database')->commit();
    return true;
}
catch (Zend_Exception $e) {
    Bootstrap::$Log->err($e->getMessage());
    Zend_Registry::get('database')->rollBack();
    return false;
}
```

Share

Improve this answer

Follow

edited Jan 30, 2018 at 13:52

Adeel

2,939 ● 7 ● 27 ● 36

answered Mar 14, 2015 at 13:09

Dens

469 ● 6 ● 7

0

In web-facing PHP, scripts are almost always invoked during a single web request. What you would really like to do in that case is start a transaction and commit it right before the script ends. If anything goes wrong, throw an exception and roll back the entire thing. Like this:

```
wrapper.php:

try {
   // start transaction
   include("your_script.php");
   // commit transaction
} catch (RollbackException $e) {
   // roll back transaction
}
```

The situation gets a little more complex with sharding, where you may be opening several connections. You have to add them to a list of connections where the transactions should be committed or rolled back at the end of the script. However, realize that in the case of sharding, unless you have a global mutex on transactions, you will not be easily able to achieve true isolation or atomicity of concurrent transactions because another script might be committing their transactions to the shards while you're committing yours. However, you might want to check out MySQL's distributed transactions.

Share Improve this answer Follow

answered Jul 15, 2013 at 1:50

Gregory Magarshak

2,019 ● 2 ● 26 ● 38

0

Use zend profiler to see begin as query text and Zend_Db_Prfiler::TRANSACTION as query type with out commit or rollback as query text afterwards. (By assuming there is no ->query("START TRANSACTION") and zend profiler enabled in your application)

Share Improve this answer Follow

answered May 15, 2014 at 7:28

siva kiran
36 ● 3

0

Maybe you can try PDO::inTransaction...returns TRUE if a transaction is currently active, and FALSE if not. I have not tested myself but it seems not bad!

Share Improve this answer Follow

answered Mar 14, 2018 at 14:20

Pepstille
1 ● 2