Transactions

In modern databases, transactions also do some other things, such as ensuring that you can’t access data  
that another person has written halfway. However, the basic idea is the same—transactions are there to  
ensure that no matter what happens, the data you work with will be in a sensible state. They guarantee  
that there will not be a situation where money is withdrawn from one account, but not deposited to  
another.

Yii2 supports a powerful transaction mechanism with savepoints.

A classic example is of transferring money from one bank account to another. To do that, you have to first  
withdraw the amount from the source account, and then deposit it to the destination account. The  
operation has to succeed in full. If you stop halfway, the money will be lost, and that is very bad. For  
instance, we have a recipient account and a sender account. We would like to transfer money from sender  
to recipient. Let’s assume that we have an account model.

Getting ready...

Our account model will be very simple and it will contain only the id and balance fields.

1. Create a new application using the Composer package manager, as described in the official guide at  
   <http://www.yiiframework.com/doc-2.0/guide-start-installation.html>.
2. Create a migration, which adds an account table, using the following command:

./yii migrate/create create\_account\_table

1. Also, update the just- created migration using the following code:

<?php

use yii\db\Schema;  
use yii\db\Migration;

class m150620\_062034\_create\_account\_table extends Migration  
{

const TABLE\_NAME = '{{%account}}';

public function up()

{

$tableOptions = null;

if ($this->db->driverName === 'mysql') {

$tableOptions = 'CHARACTER SET utf8 COLLATE utf8\_general\_ci  
ENGINE=InnoDB';

}

$this->createTable(self::TABLE\_NAME, [

'id' => Schema::TYPE\_PK,

'balance' => ' NUMERIC(15,2) DEFAULT NULL',

], $tableOptions);

}

public function down()

{

$this->dropTable(self::TABLE\_NAME);

}

}

1. Then, install migration with the following command:

./yii migrate up

1. Use Gii to create a model for the account table.
2. Create a migration, which adds some test Account models with balance for our table:  
   ./yii migrate/create add\_account\_records
3. Also, update the just-created migration using the following code:

<?php

use yii\db\Migration;  
use app\models\Account;

class m150620\_063252\_add\_account\_records extends Migration  
{

public function up()

{

$accountFirst = new Account();

$accountFirst->balance = 1110;

$accountFirst->save();

$accountSecond = new Account();

$accountSecond->balance = 779;

$accountSecond->save();

$accountThird = new Account();

$accountThird->balance = 568;

$accountThird->save();  
return true;

}

public function down()

{

$this->truncateTable('{{%account}}');  
return false;

}

}

How to do it...

1. Add the following rule to the rules method, to models/Account. php:

public function rules()

{

return [

//..

[['balance'], 'number', 'min' => 0],

//..

];

}

1. Let us assume that our balance may be only positive and that it can’t be negative.
2. Create TestController with success and error actions:

<?php

namespace app\controllers;

use app\models\Account;  
use Yii;

use yii\db\Exception;  
use yii\helpers\Html;  
use yii\helpers\VarDumper;  
use yii\web\Controller;

class TestController extends Controller  
{

public function actionSuccess()

{

$transaction = Yii::$app->db->beginTransaction();  
try {

$recipient = Account::findOne(1);

$sender = Account::findOne(2);

$transferAmount = 177;

$recipient->balance += $transferAmount;

$sender->balance -= $transferAmount;

if ($sender->save() && $recipient->save()) {

$transaction->commit();

return $this->renderContent(

Html::tag('h1', 'Money transfer was successfully')

);

} else {

$transaction->rollBack();

throw new Exception('Money transfer failed:' .  
VarDumper::dumpAsString($sender->getErrors()) .  
VarDumper::dumpAsString($recipient->getErrors())

);

}

} catch ( Exception $e ) {

$transaction->rollBack();  
throw $e;

}

}

public function actionError()

{

$transaction = Yii::$app->db->beginTransaction();  
try {

$recipient = Account::findOne(1);

$sender = Account::findOne(3);

$transferAmount = 1000;

$recipient->balance += $transferAmount;

$sender->balance -= $transferAmount;

if ($sender->save() && $recipient->save()) {

$transaction->commit();

return $this->renderContent(

Html::tag('h1', 'Money transfer was successfully')

);

} else {

$transaction->rollBack();

throw new Exception('Money transfer failed: ' .  
VarDumper::dumpAsString($sender->getErrors()) .  
VarDumper::dumpAsString($recipient->getErrors())

);

}

} catch ( Exception $e ) {

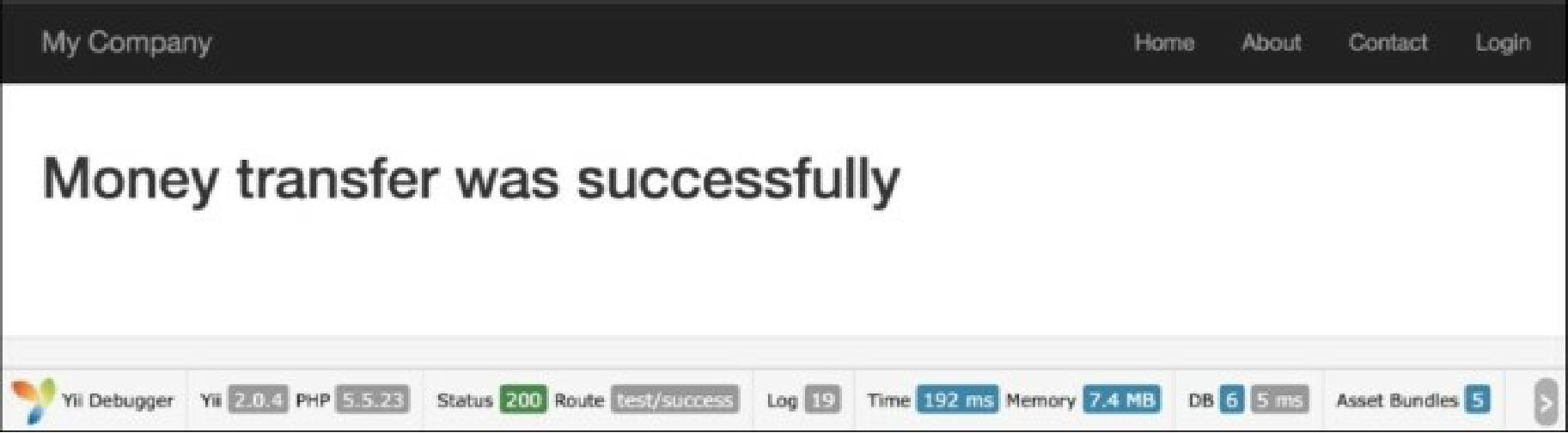
$transaction->rollBack();  
throw $e;

}

}

}

4. Run test/success and you should get the output shown in the following screenshot:



1. In this case, the transaction mechanism will not update the recipient and sender balance if some  
   error occurred.
2. Run test/error and you should get the output shown in the following screenshot:



As you will remember, we added a rule to the Account model, so our account balance can be only positive.  
The transaction will roll back in this case and it prevents a situation where money is withdrawn from a  
sender’s account but not deposited to the recipient’s account.

See also

For further information, refer to:

* [http://www.yiiframework.eom/doc-2.0/guide-db-dao.html#performing-transactions](http://www.yiiframework.com/doc-2.0/guide-db-dao.html%23performing-transactions)
* [http://www.viiframework.com/doc-2.0/guide-db-dao.html#nesting-transactions](http://www.yiiframework.com/doc-2.0/guide-db-dao.html%23nesting-transactions)