Getting data from a database

Most applications use databases today. Be it a small website or a social network, at least some parts are powered by databases.

Yii introduces three ways to allow you to work with databases. They are as follows:

* Active Record
* Query Builder
* SQL via DAO

We will use all these methods to get data from the film, film\_actor, and actor tables and show it in a list. Also, we will compare the execution time and memory usage to determine in which cases we should use these methods.

Getting ready

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. Download the Sakila database from [http://dev.mysql.com/doc/inHey-other.html](http://dev.mysql.com/doc/index-other.html).
3. Execute the downloaded SQLs; first schema, then data.
4. Configure the DB connection in config/main. php to use the Sakila database.
5. Use Gii to create models for the actor and film tables.

How to do it...

1. Create app/controllers/DbController.php as follows:

<?php

namespace app\controllers;

use app\models\Actor; use Yii;

use yii\db\Query; use yii\helpers\ArrayHelper; use yii\helpers\Html; use yii\web\Controller;

/[[1]](#footnote-1) [[2]](#footnote-2)

* Class DbController
* @package app\controllers \*/

class DbController extends Controller {

/\*\*

->j oinWith('films')

->orderBy('actor.first\_name, actor.last\_name, film.title') ->all();

return $this->renderRecords($records);

}

/[[3]](#footnote-3)

* Example of Query class usage.

\*

* @return string \*/

public function actionQuery()

{

$rows = (new Query())

->from('actor')

->innerJoin('film\_actor', 'actor.actor\_id=film\_actor.actor\_id') ->leftJoin('film', 'film.film\_id=film\_actor.film\_id')

->orderBy('actor.first\_name, actor.last\_name, actor.actor\_id,

film.title')

->all();

return $this->renderRows($rows);

}

/\*\*

* Example of SQL execution usage.

\*

* @return string \*/

public function actionSql()

{

$sql = 'SELECT [[4]](#footnote-4)

FROM actor a

JOIN film\_actor fa ON fa.actor\_id = a.actor\_id JOIN film f ON fa.film\_id = f.film\_id

ORDER BY a.first\_name, a.last\_name, a.actor\_id, f.title';

$rows = Yii::$app->db->createCommand($sql)->queryAll(); return $this->renderRows($rows);

}

/\*\*

* Render records for Active Record array.

\*

return $this->renderContent(Html::ol($items, [

'encode' => false,

]));

}

/\*\*

* Render rows for result of query.

\*

* @param array $rows

\*

\* @return string \*/

protected function renderRows(array $rows = [])

{

if (!$rows) {

return $this->renderContent('Actor list is empty.');

}

$items = [];

$films = [];

$actorId = null;

$actorName = null;

$actorFilms = null;

$lastActorId = $rows[0]['actor\_id'];

foreach ($rows as $row) {

$actorId = $row['actor\_id'];

$films[] = $row['title'];

if ($actorId != $lastActorId) {

$actorName = $row['first\_name'].' '.$row['last\_name']; $actorFilms = $films ? Html::ol($films) : null;

$items[] = $actorName.$actorFilms;

$films = [];

$lastActorId = $actorId;

}

}

if ($actorId == $lastActorId) {

$actorFilms = $films ? Html::ol($films) : null;

$items[] = $actorName.$actorFilms;

}

return $this->renderContent(Html::ol($items, [

'encode' => false,

]));

}

}

1. Here, we have three actions corresponding to the three different methods of getting data from a database.
2. After running the preceding db/ar, db/query and db/sql actions, you should get a tree showing 200 actors and 1,000 films they have acted in, as shown in the following screenshot:



1. At the bottom, there are statistics that give information about the memory usage and execution time. Absolute numbers can be different if you run this code, but the difference between the methods used should be about the same:

|  |  |  |
| --- | --- | --- |
| Method | Memory usage (megabytes) | Execution time (seconds) |
| Active Record | 21.4 | 2.398 |
| Query Builder | 28.3 | 0.477 |
| SQL (DAO) | 27.6 | 0.481 |

How it works...

The actionAr action method gets model instances using the Active Record approach.

We start with the Actor model generated with Gii to get all the actors, and specify joinwith => 'films' to get the corresponding films using a single query or eager loading through relation, which Gii builds for us from innoDB table foreign keys. We then simply iterate over all the actors and for each actor, over each film. Then, for each item, we print its name.

The actionQuery function uses Query Builder. First, we create a query for the current DB connection with \yii\db\Query. We then add query parts one by one with from, joininner, and leftJoin. These methods escape values, tables, and field names automatically. The all() function of \yii\db\Query returns an array of raw database rows. Each row is also an array, indexed with result field names. We pass the result to renderRows, which renders it.

With actionSql, we do the same, except that we pass SQL directly instead of adding its parts one by one. It’s worth mentioning that we should escape parameter values manually using Yii: :app()->db- >quotevalue before using them in the query string:

The renderRows method renders the Query Builder.

The renderRecords method renders the active records.

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Active Record | Query Builder | SQL (DAO) |
| Syntax | This will do SQL for you.  Gii will generate models and relations for you.  Works with models, completely OO-style, and a very clean API.  Produces an array of properly nested models as the result. | Clean API, suitable for building query on the fly.  Produces raw data arrays as the result. | Good for complex SQL.  Manual values and keyword quoting.  Not very suitable for building a query on the fly.  Produces raw data arrays as the result. |
| Performance | Higher memory usage and execution time compared to SQL and Query Builder. | Okay. | Okay. |
| Extra  features | Quotes values and names automatically.  Behaviors. Before/after hooks. Validation. Prototyping selects. | Quotes values and names automatically. | None. |
| Best for | Update, delete, and create actions for single models (the model gives a huge benefit when using with forms). | Working with large amount of data and building queries on the fly. | Complex queries you want to complete with pure SQL and have maximum possible performance. |

There’s more...

In order to learn more about working with databases in Yii, refer to the following resources:

* [http://www.yiiframework.eom/doc-2.0/guide-db-dao.html](http://www.yiiframework.com/doc-2.0/guide-db-dao.html)
* <http://www.yiiframework.com/doc-2.0/guide-db-query-builder.html>
* [http://www.yiiframework.com/doc-2.0/guide-db-active-rerord.html](http://www.yiiframework.com/doc-2.0/guide-db-active-record.html)

1. Example of Active Record usage.

   \*

   \* @return string \*/

   public function actionAr() [↑](#footnote-ref-1)
2. {

   $records = Actor::find() [↑](#footnote-ref-2)
3. {

   if ( !$records) {

   return $this->renderContent('Actor list is empty.');

   }

   $items = [];

   foreach ($records as $record) {

   $actorFilms = $record->films

   ? Html::ol(ArrayHelper::getColumn($record->films, 'title')): null; $actorName = $record->first\_name.' '.$record->last\_name;

   $items[] = $actorName.$actorFilms;

   } [↑](#footnote-ref-3)
4. @param array $records

   \*

   \* @return string \*/

   protected function renderRecords(array $records = []) [↑](#footnote-ref-4)