Working with Data

Tests should not affect each other. That's a rule of thumb. When tests interact with a database, they may change the data inside it, which would eventually lead to data inconsistency. A test may try to insert a record that has already been inserted, or retrieve a deleted record. To avoid test failures, the database should be brought back to its initial state before each test. Codeception has different methods and approaches to get your data cleaned.

This chapter summarizes all of the notices on clean-ups from the previous chapters and suggests the best strategies of how to choose data storage backends.

When we decide to clean up a database, we should make this cleaning as fast as possible. Tests should always run fast. Rebuilding the database from scratch is not the best way, but might be the only one. In any case, you should use a special test database for testing. **Do not ever run tests on development or production databases!**

Db

Codeception has a `Db` module, which takes on most of the tasks of database interaction.

```
modules:
    config:
        Db:
            dsn: 'PDO DSN HERE'
            user: 'root'
            password:
```

> Use module parameters (http://codeception.com/docs/06-ModulesAndHelpers#Dynamic-Configuration-With-Params) to set the database credentials from environment variables or from application configuration files.

Db module can cleanup database between tests by loading a database dump. This can be done by parsing SQL file and executing its commands using current connection

```
modules:
    config:
        Db:
            dsn: 'PDO DSN HERE'
            user: 'root'
            password:
            dump: tests/_data/your-dump-name.sql
            cleanup: true # reload dump between tests
            populate: true # load dump before all tests
```

Alternatively an external tool (like mysql client, or pg_restore) can be used. This approach is faster and won't produce parsing errors while loading a dump. Use `populator` config option to specify the command. For MySQL it can look like this:

```
modules:
   enabled:
      - Db:
         dsn: 'mysql:host=localhost;dbname=testdb'
         user: 'root'
         password: ''
         cleanup: true # run populator before each test
         populate: true # run populator before all test
         populator: 'mysql -u $user $dbname < tests/_data/dump.sql'
```

See the Db module reference (http://codeception.com/docs/modules/Db#SQL-data-dump) for more examples.

To ensure database dump is loaded before all tests add `populate: true`. To clean current database and reload dump between tests use `cleanup: true`.

> A full database clean-up can be painfully slow if you use large database dumps. It is recommended to do more data testing on the functional and integration levels, this way you can get performance bonuses from using ORM.

In acceptance tests, your tests are interacting with the application through a web server. This means that the test and the application work with the same database. You should provide the same credentials in the Db module that your application uses, then you can access the database for assertions (`seeInDatabase` actions) and to perform automatic clean-ups.

The Db module provides actions to create and verify data inside a database.

If you want to create a special database record for one test, you can use `haveInDatabase` (http://codeception.com/docs/modules/Db#haveInDatabase) method of `Db` module:

```
<?php
$I->haveInDatabase('posts', [
  'title' => 'Top 10 Testing Frameworks',
  'body' => '1. Codeception'
]);
$I->amOnPage('/posts');
$I->see('Top 10 Testing Frameworks');
```

`haveInDatabase` inserts a row with the provided values into the database. All added records will be deleted at the end of the test.

If you want to check that a table record was created use `seeInDatabase` (http://codeception.com/docs/modules/Db#haveInDatabase) method:

```
<?php
$I->amOnPage('/posts/1');
$I->fillField('comment', 'This is nice!');
$I->click('Submit');
$I->seeInDatabase('comments', ['body' => 'This is nice!']);
```

See the module reference (http://codeception.com/docs/modules/Db) for other methods you can use for database testing.

There are also modules for MongoDb (http://codeception.com/docs/modules/MongoDb), Redis (http://codeception.com/docs/modules/Redis), and Memcache (http://codeception.com/docs/modules/Memcache) which behave in a similar manner.

Sequence

If the database clean-up takes too long, you can follow a different strategy: create new data for each test. This way, the only problem you might face is duplication of data records. Sequence (http://codeception.com/docs/modules/Sequence) was created to solve this. It provides the `sq()` function which generates unique suffixes for creating data in tests.

ORM modules

Your application is most likely using object-relational mapping (ORM) to work with the database. In this case, Codeception allows you to use the ORM methods to work with the database, instead of accessing the database directly. This way you can work with models and entities of a domain, and not on tables and rows.

By using ORM in functional and integration tests, you can also improve performance of your tests. Instead of cleaning up the database after each test, the ORM module will wrap all the database actions into transactions and roll it back at the end. This way, no actual data will be written to the database. This clean-up strategy is enabled by default, you can disable it by setting `cleanup: false` in the configuration of any ORM module.

ActiveRecord

Popular frameworks like Laravel, Yii, and Phalcon include an ActiveRecord data layer by default. Because of this tight integration, you just need to enable the framework module, and use its configuration for database access.

Corresponding framework modules provide similar methods for ORM access:

- haveRecord
- seeRecord
- dontSeeRecord
- grabRecord

They allow you to create and check data by model name and field names in the model. Here is the example in Laravel:

```
<?php
// create record and get its id
$id = $I->haveRecord('posts', ['body' => 'My first blogpost', 'user_id' => 1]);
$I->amOnPage('/posts/'.$id);
$I->see('My first blogpost', 'article');
// check record exists
$I->seeRecord('posts', ['id' => $id]);
$I->click('Delete');
// record was deleted
$I->dontSeeRecord('posts', ['id' => $id]);
```

> Laravel5 module also provides `haveModel`, `makeModel` methods which use factories to generate models with fake data.

If you want to use ORM for integration testing only, you should enable the framework module with only the `ORM` part enabled:

```
modules:
    enabled:
        - Laravel5:
            - part: ORM
```

```
modules:
    enabled:
        - Yii2:
            - part: ORM
```

This way no web actions will be added to `$I` object.

If you want to use ORM to work with data inside acceptance tests, you should also include only the ORM part of a module. Please note that inside acceptance tests, web applications work inside a webserver, so any test data can't be cleaned up by rolling back transactions. You will need to disable cleaning up, and use the `Db` module to clean the database up between tests. Here is a sample config:

```
modules:
    enabled:
        - WebDriver:
            url: http://localhost
            browser: firefox
        - Laravel5:
            cleanup: false
        - Db
```

DataMapper

Doctrine is also a popular ORM, unlike some others it implements the DataMapper pattern and is not bound to any framework. The Doctrine2 (http://codeception.com/docs/modules/Doctrine2) module requires an `EntityManager` instance to work with. It can be obtained from a Symfony framework or Zend Framework (configured with Doctrine):

```
modules:
    enabled:
        - Symfony
        - Doctrine2:
            depends: Symfony
```

```
modules:
    enabled:
        - ZF2
        - Doctrine2:
            depends: ZF2
```

If no framework is used with Doctrine you should provide the `connection_callback` option with a valid callback to a function which returns an `EntityManager` instance.

Doctrine2 also provides methods to create and check data:

- `haveInRepository`
- `grabFromRepository`
- `grabEntitiesFromRepository`
- `seeInRepository`
- `dontSeeInRepository`

DataFactory

Preparing data for testing is a very creative, although boring, task. If you create a record, you need to fill in all the fields of the model. It is much easier to use Faker (https://github.com/fzaninotto/Faker) for this task, which is more effective to set up data generation rules for models. Such a set of rules is called *factories* and are provided by the DataFactory (http://codeception.com/docs/modules/DataFactory) module.

Once configured, it can create records with ease:

```
<?php
// creates a new user
$user_id = $I->have('App\Model\User');
// creates 3 posts
$I->haveMultiple('App\Model\Post', 3);
```

Created records will be deleted at the end of a test. The DataFactory module only works with ORM, so it requires one of the ORM modules to be enabled:

```
modules:
    enabled:
        - Yii2:
            configFile: path/to/config.php
        - DataFactory:
            depends: Yii2
```

```
modules:
    enabled:
        - Symfony
        - Doctrine2:
            depends: Symfony
        - DataFactory:
            depends: Doctrine2
```

DataFactory provides a powerful solution for managing data in integration/functional/acceptance tests. Read the full reference (http://codeception.com/docs/modules/DataFactory) to learn how to set this module up.

Conclusion

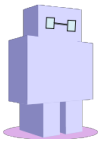
Codeception also assists the developer when dealing with data. Tools for database population and cleaning up are bundled within the `Db` module. If you use ORM, you can use one of the provided framework modules to operate with database through a data abstraction layer, and use the DataFactory module to generate new records with ease.

- **Next Chapter: WebServices > (/docs/10-WebServices)**
- **Previous Chapter: < Customization (/docs/08-Customization)**

Looking for commercial support? Request official consulting service (http://sdclabs.com/codeception?utm_source=codeception.com&utm_medium=docs_bottom&utm_term=link&utm_campaign=reference)

- Credits (/credits)
- Releases (/changelog)
- Commercial Services (http://sdclabs.com/codeception?utm_source=codeception.com&utm_medium=bottom_menu&utm_term=link&utm_campaign=reference)
- License (https://github.com/Codeception/Codeception/blob/master/LICENSE)

Codeception Family



Robo

(http://robo.li)(http://robo.li)
Modern PHP **Task Runner**. Allows to declare tasks with zero configuration in pure PHP.

CodeceptJS

(http://codecept.io)(http://codecept.io)
Codeception for **NodeJS**. Write acceptance tests in ES6 and execute in webdriverio, Selenium WebDriver, and Protractor.