**Db**

Access a database.

The most important function of this module is to clean a database before each test. This module also provides actions to perform checks in a database, e.g. [seeInDatabase()](http://codeception.com/docs/modules/Db#seeInDatabase)

In order to have your database populated with data you need a raw SQL dump. Simply put the dump in the tests/\_data directory (by default) and specify the path in the config. The next time after the database is cleared, all your data will be restored from the dump. Don’t forget to include CREATE TABLE statements in the dump.

Supported and tested databases are:

* MySQL
* SQLite (i.e. just one file)
* PostgreSQL

Also available:

* MS SQL
* Oracle

Connection is done by database Drivers, which are stored in the Codeception\Lib\Driver namespace. [Check out the drivers](https://github.com/Codeception/Codeception/tree/2.3/src/Codeception/Lib/Driver) if you run into problems loading dumps and cleaning databases.

**Config**

* dsn required - PDO DSN
* user required - username to access database
* password required - password
* dump - path to database dump
* populate: false - whether the the dump should be loaded before the test suite is started
* cleanup: false - whether the dump should be reloaded before each test
* reconnect: false - whether the module should reconnect to the database before each test
* ssl\_key - path to the SSL key (MySQL specific, @see http://php.net/manual/de/ref.pdo-mysql.php#pdo.constants.mysql-attr-key)
* ssl\_cert - path to the SSL certificate (MySQL specific, @see http://php.net/manual/de/ref.pdo-mysql.php#pdo.constants.mysql-attr-ssl-cert)
* ssl\_ca - path to the SSL certificate authority (MySQL specific, @see http://php.net/manual/de/ref.pdo-mysql.php#pdo.constants.mysql-attr-ssl-ca)

**Example**

modules:

enabled:

- Db:

dsn: 'mysql:host=localhost;dbname=testdb'

user: 'root'

password: ''

dump: 'tests/\_data/dump.sql'

populate: true

cleanup: true

reconnect: true

ssl\_key: '/path/to/client-key.pem'

ssl\_cert: '/path/to/client-cert.pem'

ssl\_ca: '/path/to/ca-cert.pem'

**SQL data dump**

There are two ways of loading the dump into your database:

**Populator**

The recommended approach is to configure a populator, an external command to load a dump. Command parameters like host, username, password, database can be obtained from the config and inserted into placeholders:

For MySQL:

modules:

enabled:

- Db:

dsn: 'mysql:host=localhost;dbname=testdb'

user: 'root'

password: ''

dump: 'tests/\_data/dump.sql'

populate: true # run populator before all tests

cleanup: true # run populator before each test

populator: 'mysql -u $user -h $host $dbname < $dump'

For PostgreSQL (using pg\_restore)

modules:

enabled:

- Db:

dsn: 'pgsql:host=localhost;dbname=testdb'

user: 'root'

password: ''

dump: 'tests/\_data/db\_backup.dump'

populate: true # run populator before all tests

cleanup: true # run populator before each test

populator: 'pg\_restore -u $user -h $host -D $dbname < $dump'

Variable names are being taken from config and DSN which has a keyword=value format, so you should expect to have a variable named as the keyword with the full value inside it.

PDO dsn elements for the supported drivers:

* MySQL: [PDO\_MYSQL DSN](https://secure.php.net/manual/en/ref.pdo-mysql.connection.php)
* SQLite: [PDO\_SQLITE DSN](https://secure.php.net/manual/en/ref.pdo-sqlite.connection.php)
* PostgreSQL: [PDO\_PGSQL DSN](https://secure.php.net/manual/en/ref.pdo-pgsql.connection.php)
* MSSQL: [PDO\_SQLSRV DSN](https://secure.php.net/manual/en/ref.pdo-sqlsrv.connection.php)
* Oracle: [PDO\_OCI DSN](https://secure.php.net/manual/en/ref.pdo-oci.connection.php)

**Dump**

Db module by itself can load SQL dump without external tools by using current database connection. This approach is system-independent, however, it is slower than using a populator and may have parsing issues (see below).

Provide a path to SQL file in dump config option:

modules:

enabled:

- Db:

dsn: 'mysql:host=localhost;dbname=testdb'

user: 'root'

password: ''

populate: true # load dump before all tests

cleanup: true # load dump for each test

dump: 'tests/\_data/dump.sql'

To parse SQL Db file, it should follow this specification:

* Comments are permitted.
* The dump.sql may contain multiline statements.
* The delimiter, a semi-colon in this case, must be on the same line as the last statement:

-- Add a few contacts to the table.

REPLACE INTO `Contacts` (`created`, `modified`, `status`, `contact`, `first`, `last`) VALUES

(NOW(), NOW(), 1, 'Bob Ross', 'Bob', 'Ross'),

(NOW(), NOW(), 1, 'Fred Flintstone', 'Fred', 'Flintstone');

-- Remove existing orders for testing.

DELETE FROM `Order`;

**Query generation**

seeInDatabase, dontSeeInDatabase, seeNumRecords, grabFromDatabase and grabNumRecords methods accept arrays as criteria. WHERE condition is generated using item key as a field name and item value as a field value.

Example:

<?php

$I->seeInDatabase('users', array('name' => 'Davert', 'email' => 'davert@mail.com'));

Will generate:

SELECT COUNT(\*) FROM `users` WHERE `name` = 'Davert' AND `email` = 'davert@mail.com'

Since version 2.1.9 it’s possible to use LIKE in a condition, as shown here:

<?php

$I->seeInDatabase('users', array('name' => 'Davert', 'email like' => 'davert%'));

Will generate:

SELECT COUNT(\*) FROM `users` WHERE `name` = 'Davert' AND `email` LIKE 'davert%'

**Public Properties**

* dbh - contains the PDO connection
* driver - contains the Connection Driver

**Actions**

**dontSeeInDatabase**

Effect is opposite to ->seeInDatabase

Asserts that there is no record with the given column values in a database. Provide table name and column values.

<?php

$I->dontSeeInDatabase('users', ['name' => 'Davert', 'email' => 'davert@mail.com']);

Fails if such user was found.

Comparison expressions can be used as well:

<?php

$I->dontSeeInDatabase('posts', ['num\_comments >=' => '0']);

$I->dontSeeInDatabase('users', ['email like' => 'miles%']);

Supported operators: <, >, >=, <=, !=, like.

* param string $table
* param array $criteria

**grabColumnFromDatabase**

Fetches all values from the column in database. Provide table name, desired column and criteria.

<?php

$mails = $I->grabColumnFromDatabase('users', 'email', array('name' => 'RebOOter'));

* param string $table
* param string $column
* param array $criteria
* return array

**grabFromDatabase**

Fetches a single column value from a database. Provide table name, desired column and criteria.

<?php

$mail = $I->grabFromDatabase('users', 'email', array('name' => 'Davert'));

Comparison expressions can be used as well:

<?php

$post = $I->grabFromDatabase('posts', ['num\_comments >=' => 100']);

$user = $I->grabFromDatabase('users', ['email like' => 'miles%']);

Supported operators: <, >, >=, <=, !=, like.

* param string $table
* param string $column
* param array $criteria

**grabNumRecords**

Returns the number of rows in a database

* param string $table Table name
* param array $criteria Search criteria [Optional]
* return int

**haveInDatabase**

Inserts an SQL record into a database. This record will be erased after the test.

<?php

$I->haveInDatabase('users', array('name' => 'miles', 'email' => 'miles@davis.com'));

?>

* param string $table
* param array $data
* return integer $id

**isPopulated**

**not documented**

**seeInDatabase**

Asserts that a row with the given column values exists. Provide table name and column values.

<?php

$I->seeInDatabase('users', ['name' => 'Davert', 'email' => 'davert@mail.com']);

Fails if no such user found.

Comparison expressions can be used as well:

<?php

$I->seeInDatabase('posts', ['num\_comments >=' => '0']);

$I->seeInDatabase('users', ['email like' => 'miles@davis.com']);

Supported operators: <, >, >=, <=, !=, like.

* param string $table
* param array $criteria

**seeNumRecords**

Asserts that the given number of records were found in the database.

<?php

$I->seeNumRecords(1, 'users', ['name' => 'davert'])

?>

* param int $expectedNumber Expected number
* param string $table Table name
* param array $criteria Search criteria [Optional]

**updateInDatabase**

Update an SQL record into a database.

<?php

$I->updateInDatabase('users', array('isAdmin' => true), array('email' => 'miles@davis.com'));

?>

* param string $table
* param array $data
* param array $criteria

Module reference is taken from the source code. [Help us to improve documentation. Edit module reference](https://github.com/Codeception/Codeception/tree/2.3/src/Codeception/Module/Db.php)

Let us help you take Codeception further. We offer a number of paid services that can help you and your organization get the most out of Codeception Enterprise. Software testing & test automation development and outsourcing. **Request a quote for** [**enterprise support**](http://sdclabs.com/codeception?utm_source=codeception.com&utm_medium=docs_bottom&utm_term=link&utm_campaign=reference) **or** [**trainings**](http://sdclabs.com/trainings?utm_source=codeception.com&utm_medium=docs_bottom&utm_term=link&utm_campaign=reference)

Codeception is a BDD-styled PHP testing framework, brought to you by [Codeception Team](http://codeception.com/credits). Logo by [Mr. Adnan](https://twitter.com/adnanblog). OpenSource **MIT Licensed**.

#### Thanks to

* [Installation](http://codeception.com/install)
* [Credits](http://codeception.com/credits)
* [Releases](http://codeception.com/changelog)
* [Enterprise Support](http://sdclabs.com/codeception?utm_source=codeception.com&utm_medium=bottom_menu&utm_term=link&utm_campaign=reference)
* [Trainings](http://sdclabs.com/trainings?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/)

Modern PHP **Task Runner**. Allows to declare tasks with zero configuration in pure PHP.

[**CodeceptJS**](http://codecept.io/)

Codeception for **NodeJS**. Write acceptance tests in ES6 and execute in webdriverio, Selenium WebDriver, and Protractor.

**© 2011–2017**

[Follow @Codeception](https://twitter.com/codeception)