pytest-django-queries Documentation

Release 1.0.0

KOCAK Mikail (NyanKiyoshi)

Sep 03, 2020

Contents

`pytest-django-queries` is pytest plugin tool for measuring the database query count of a django project. It captures the SQL queries of marked tests to generate reports from them that can then be analyzed, proceeded and even integrated to CIs and GitHub as a peer reviewing tool (bot).

This is used to detect and correct features that are missing optimizations or that should be rethought, and which parts are doing great.

This is also used to quickly see and compare differences of made changes through the included diff tool.

This tool supports the Python versions: 2.7, 3.4, 3.5, 3.6, 3.7, and 3.8.

CHAPTER 1

Quick Start

1. Install the tool by running `pip install pytest-django-queries`;
2. Use the plugin by marking tests or by using the provided fixture:

```
import pytest


@pytest.mark.count_queries
def test_query_performances():
    Model.objects.all()


# Or...
def test_another_query_performances(count_queries):
    Model.objects.all()
```

3. Run `pytest`;
4. Then use `django-queries show` to show the results directly into your console:

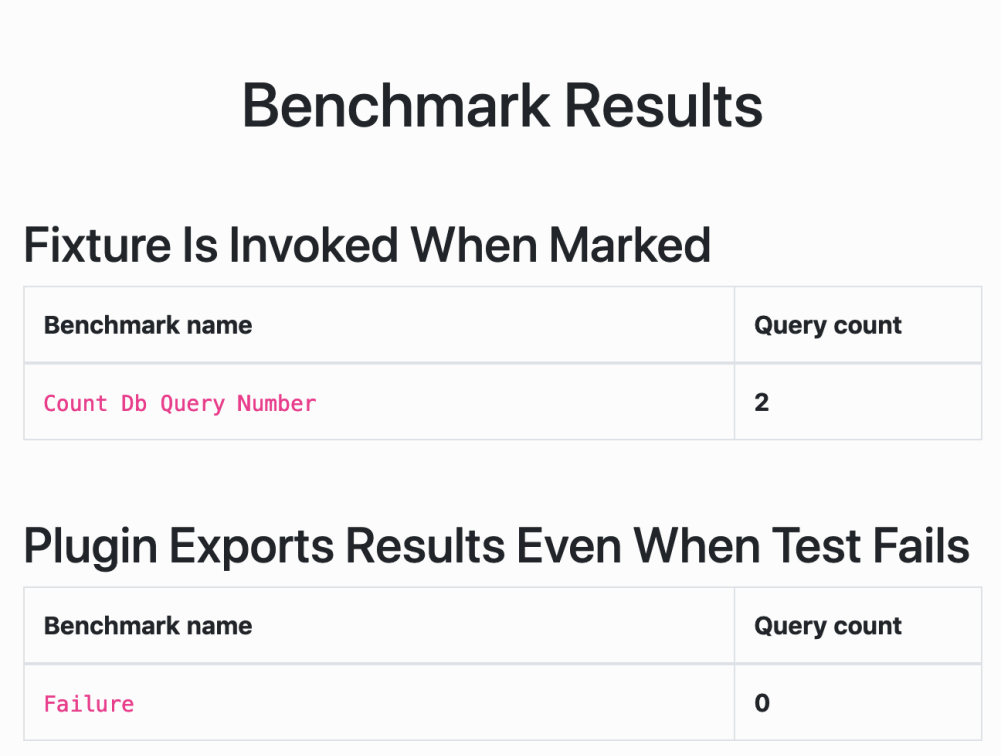
```
+---------+-------------------------+
| Module  |          Tests          |
+---------+-------------------------+
| module1 | +-----------+---------+ |
|         | | Test Name | Queries | |
|         | +-----------+---------+ |
|         | |   test1   |    0    | |
|         | +-----------+---------+ |
|         | |   test2   |    1    | |
|         | +-----------+---------+ |
+---------+-------------------------+
| module2 | +-----------+---------+ |
|         | | Test Name | Queries | |
|         | +-----------+---------+ |
|         | |   test1   |   123   | |
```

(continues on next page)

(continued from previous page)

```
|          | +-----------+---------+ |
+----------+-------------------------+
| module3  |                         |
+----------+-------------------------+
```

5. Or for a nicer presentation, use `django-queries html` to export the results as HTML. See this example for a demo!



Benchmark Results

Fixture Is Invoked When Marked

Benchmark name	Query count
Count Db Query Number	2

Plugin Exports Results Even When Test Fails

Benchmark name	Query count
Failure	0

6. By running it twice with the option described *here* and by running `django-queries diff` you will get something like this:

```
# another module
  test name                left count      right count
  ---------------------    -----------     -----------
+ new test                             -               1

# module
  test name                left count      right count
  ---------------------    -----------     -----------
- degraded func                      15              16
+ improved func                      20              19
  unchanged func                      1               1
```

Warning: Please take a quick look at our *recommendations* before starting using the plugin in your project tests.

CHAPTER 2

Getting Help

Feel free to open an issue in our GitHub repository! Or reach hello@vanille.bid.

CHAPTER 3

More Topics

3.1 Recommendations (Must Read!)

3.1.1 Fixtures That Generate Queries

If your test case is using fixtures that are generating any database queries, you will end up with unwanted queries being counted in your tests. For that reason, we recommend you to manually request the usage of the `count_queries` fixture and put it as the last parameter of your test.

By doing so, you will be sure that the query counter is actually always executed last and does not wrap any other fixtures.

Along side, you might want to still use the plugin's `count_queries` marker which is useful to keep your tests separated from the query counting tests.

Your code will look like something like this:

```
import pytest


@pytest.mark.count_queries(autouse=False)
def test_retrieve_main_menu(fixture_making_queries, count_queries):
    pass
```

3.1.2 Using `pytest-django` Alongside of Counting Queries

You are most likely using the `pytest-django` plugin which is really useful for django testing. By following the previous section's example, you might want to unblock the test database as well. You would do something like this:

```
import pytest
```

(continues on next page)

(continued from previous page)

```
@pytest.mark.django_db
@pytest.mark.count_queries(autouse=False)
def test_retrieve_main_menu(any_fixture, other_fixture, count_queries):
    pass
```

3.2 The Diff Command

The plugin can backup the test results for you if you run the `django-queries backup [BACKUP_PATH]` command. It will create a backup to `.pytest-query.old` by default if previous results were found.

Warning: Bear in mind that it will override any existing backup file in the provided or default path.

After running `pytest` again, you can run `django-queries diff` to show the changes. Make sure you actually had previous results, otherwise it will have nothing to compare.

3.3 Customizing the Outputs

3.3.1 Customizing the console output

To be done, feel free to open a PR!

3.3.2 Customizing the HTML output

We are using jinja2 as a template engine for rendering results. You can customize it by passing the `--template <YOUR_TEMPLATE_PATH>` option.

The test data are passed to the `data` variable. It contains an iterator of:

```
[
    ("the module name", [
        "the test name", {"query-count": int},
        ...
    ],
    ...
]
```

Note: We also provide a `humanize` function that takes a string a removes from it the `test_` prefix.

For example, you would do the following to show all the results:

```
{% for module_name, module_data in data %}
    <section>
        <h2 class="text-capitalize">{{ humanize(module_name) }}</h2>

        <table class="table table-bordered mb-5">
            <thead>
```

(continues on next page)

(continued from previous page)

```
                <tr>
                    <th>Benchmark name</th>
                    <th>Query count</th>
                </tr>
            </thead>

            <tbody>
                {% for test_entry in module_data %}
                    <tr>
                        <td class="text-capitalize">
                            <code>{{ humanize(test_entry.test_name) }}</code>
                        </td>
                        <td>
                            <strong>{{ test_entry['query-count'] }}</strong>
                        </td>
                    </tr>
                {% else %}
                    <tr>
                        <td colspan="2">
                            <p>No data.</p>
                        </td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    </section>
{% else %}
    <p>No data.</p>
{% endfor %}
```

3.4 Plugin Usage

The plugin supports some optional parameters that are defined below.

3.4.1 Customizing the Save Path

```
--django-db-bench=PATH
Output file for storing the results. Default: .pytest-queries
```

3.4.2 Backing Up Results

The easiest way is to run the `django-queries backup` command which will create a copy of the current results.

Another way is by passing the `--django-backup-queries` parameter to backup previous results to *.pytest-django.old*‘.

Or pass a custom path.

```
--django-backup-queries=[PATH]
  Whether the old results should be backed up or not before overriding.
```

3.4.3 Running Tests Separately

To only run the `count_queries` marked tests and nothing else, you can run `pytest -v -m count_queries`.

3.5 CLI Usage

```
Usage: django-queries [OPTIONS] COMMAND [ARGS]...

  Command line tool for pytest-django-queries.

Options:
  --help  Show this message and exit.

Commands:
  html  Render the results as HTML instead of a raw table.
  show  View a given rapport.
```

3.5.1 The HTML Command

```
Usage: django-queries html [OPTIONS] [INPUT_FILE] [-o OUTPUT FILE]

Render the results as HTML instead of a raw table.

Options:
    -o                      The path to save the HTML file into
                            django-queries.html by default.
                            You can pass a dash (-) to write to stdout as well.

    --template JINJA2_FILE  Use a custom jinja2 template for rendering HTML results.

    --help                  Show this message and exit.
```

3.5.2 The SHOW Command

```
Usage: django-queries show [OPTIONS] [INPUT_FILE]

View a given rapport.

Options: none
```

3.5.3 The DIFF Command

```
Usage: django-queries diff [OPTIONS] [LEFT_FILE] [RIGHT_FILE]

Render the diff as a console table with colors.

Options: none
```

More details on how to use the diff command properly.

3.6 Contributing

To contribute, you can fork us! And open any pull request or tackle any issue in our GitHub repository.

3.6.1 Recommendations

- Try to be on one of the latest master versions.
- Try to always put and commit your change sets into a new and meaningful branch in your fork.
- Update the changelog file with the changes you made.

3.6.2 Code Style

We are using Black and Flake 8 to ensure a consistent code-style and reduce common Python issues in changes.

You can install a checker by running `pre-commit install` after installing our development requirements (`pip install -e '.[dev]'`). After that, you can add your changes through `git` and run `pre-commit` to check if your changes are issue-free.

3.6.3 Pull Request Review Process

Your contributions will get reviewed and will receive comments, remarks and suggestion to get the best of your pull request! It may take time, days or even weeks before you get a review from us. But don't worry, we won't forget about it, it just mean it is in a backlog because we are too busy for now.

You will get reviews from bots (CIs) that will succeed or fail. Mostly from travis-ci and codecov. Please be careful about them, they are important checks that we get your contribution denied as long as those checks are not passing.

Travis-ci is there to check that your changes work (tests and linters). If travis fails, it means something is wrong with your changes. Look at the logs, it will tell you what's going on!

Codecov is there to report the test coverage of your changes. We have a strict 100% coverage, meaning that all the code is covered by automatic tests. Please test all your changes and test them hastily, don't test just for the sake of testing and to get a proper coverage... it's wrong. We want the tests to prevent any error and any potential breaking from changes!

Finally, make sure you are using the latest version of the dependencies and that you have read our documentations.

3.7 Changelog

3.7.1 v1.0.0 - June 7th 2019

- Released the stable v1.0.0 release without any changes.

3.7.2 v1.0.0rc3 - June 6th 2019

- Added support for running tests into multiple workers (`pytest-xdist`).

3.7.3 v1.0.0rc2 - June 5th 2019

- Renamed the marker description to be more meaningful about was it does.
- Fixed a typo in the project description (PyPi and GitHub).
- Added help texts for named parameters in the cli.
- Fixed the wrong help text saying it is taking an integer when it actually expects a file path.
- Users can now mark tests without having the `count_queries` fixture injected automatically if a custom order or manual usage is needed.
- Added a better filtering of unwanted keywords in humanization of test names. It now handles test cases names inside modules (dotted import names).
- Added a `backup` command to `django-queries` to make it easier of making a copy of the current results.

3.7.4 v1.0.0rc1 - May 24th 2019

- Users can now backup/copy their previous results using the `--django-backup-queries` parameter when running `pytest`.
- The HTML cli command now exports to `django-queries-results.html` by default instead of `stdout`, instead, users have to run `django-queries html -` for the output to go in `stdout`.
- The code now enforces the Black code style and Flake 8 checks in addition to isort.

3.7.5 v1.0.0b1 - May 24th 2019

- Implement a `diff` command for comparing results.

3.7.6 v1.0.0a2 - May 17th 2019

- The requirements that could generate any diverging results between installation have now been freeze.
- A "Read The Docs" documentation has been made and published.
- Guides on how to release and contribute have been added.
- The HTML template has been moved to its own file under the package directory as `templates/default_bootstrap.jinja2`.
- The Cli commands are now taking optionally the report path file, so it can now be omitted.

3.7.7 v1.0.0a1 - May 13th 2019

- In #12, stopped storing the benchmark results in a file named after the current date and time. Instead, it will always save into `.django-queries` and won't contain a `json` file extension anymore to make it less appealing as it's not meant to be read by a human.
- In #12, dropped the environment variable `PYTEST_QUERIES_SAVE_PATH` and replaced and introduced the `--django-db-bench PATH` option instead, which does exactly the same thing.

3.7.8 v1.0.0.dev1 - May 12th 2019

- Introduced the cli (#3) with two commands:
  - `show` that process a given benchmark result to render a summary table
  - `html` render the table in HTML, the template can be customized using `--template <path>`

3.7.9 v0.1.0 - May 7th 2019

- The plugin is now able to support multiple pytest sessions without conflicting (#1)
- The plugin non-longer benchmarks everything but instead, the user is charged to manually flag each test as to be or not to be benchmarked (#1).

3.7.10 v0.0.0 - May 5th 2019

- Initial demo release.