Django North Field Documentation

Release 0.2.7.dev0

Lauréline Guérin

Feb 18, 2020

Contents

Contents:

CHAPTER 1

Django North

Django library for managing and executing hand-written PostgreSQL migrations.

Let your favorite DBAs define the database schema, and provide blue/green migration files. Drop django native migrations, and use DBA's migrations everywhere.

1.1 Requirements

- **Postgresql only** (>= 9.4)
- Django, obviously. v1.11, v2.0, v2.1
- Running under Python 2.7, 3.5, 3.6 or 3.7

1.2 Documentation

The full documentation is at https://django-north.readthedocs.org.

1.3 Quickstart

Install Django North:

```
pip install django-north
```

In your `settings.py`:

```
INSTALLED_APPS = [
    # ...
    "django_north",
]

NORTH_MANAGE_DB = True
NORTH_MIGRATIONS_ROOT = '/path/to/sql/migrations/'
NORTH_TARGET_VERSION = '1.42'
```

1.4 Running Tests

You will need a usable Postgresql database in order to test the project. For example:

```
source <YOURVIRTUALENV>/bin/activate
export DATABASE_URL=postgres://USER:PASSWORD@HOST:PORT/NAME
(myenv) $ pip install -r requirements_test.txt
```

Run tests for a specific version

```
(myenv) $ ./runtest
```

Run tests for all versions (if tox is installed globally, you don't need a virtual environment)

```
$ tox
```

1.5 Using the project

Many operations are documented in the Makefile. For more information, use:

```
$ make help
```

1.6 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-north
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-north
$ pip install django-north
```

CHAPTER 3

Usage

3.1 Configuration

In your `settings.py` :

```
INSTALLED_APPS = [
    # ...
    "django_north",
]

NORTH_MANAGE_DB = True
NORTH_MIGRATIONS_ROOT = '/path/to/sql/migrations/'
NORTH_TARGET_VERSION = '1.42'
```

List of available settings:

- `NORTH_MANAGE_DB`: if `True`, the database will be managed by north. Default value `False`
- `NORTH_MIGRATIONS_ROOT`: a path to your migration repository. **required**
- `NORTH_TARGET_VERSION`: the target SQL version (the version needed for your codebase). **required**
- `NORTH_SCHEMA_VERSION`: optional. To be used if you need to force the version used to init a new DB.
- `NORTH_SCHEMA_TPL`: default value `schema_{}.sql`
- `NORTH_FIXTURES_TPL`: default value `fixtures_{}.sql`
- `NORTH_ADDITIONAL_SCHEMA_FILES`: **deprecated** list of sql files to load before the schema. For example: a file of DB roles, some extensions. Default value: `[]`
- `NORTH_BEFORE_SCHEMA_FILES`: list of sql files, dirs and globs to load before the schema. If it's a dir or a glob, load only sql files in alphabetical order. For example: a file of DB roles, some extensions. Default value: `[]`
- `NORTH_AFTER_SCHEMA_FILES`: list of sql files, dirs and globs to load after the schema. If it's a dir or a glob, load only sql files in alphabetical order. For example: a file of permissions, grants on tables Default value: `[]`

- NORTH_CURRENT_VERSION_DETECTOR: the current version detector. Default value: django_north.management.migrations.get_current_version_from_table
- NORTH_NON_TRANSACTIONAL_KEYWORDS: list of keywords. If a keyword is found in a SQL non manual file, the file will always be run SQL instruction by SQL instruction. Else, a non manual file is run in a single execute call. Default value: ['CONCURRENTLY', 'ALTER TYPE', 'VACUUM']
- NORTH_DISCARD_ALL: when set to True (this is the default value), will run a DISCARD ALL; SQL statement after each executed file. More on the root cause here Issue #33 on Github (TL;DR: for security reasons, it's better to run this to reset all variables previously set, especially the search_path).

In production environments, NORTH_MANAGE_DB should be disabled, because the database is managed directly by the DBA team (database as a service).

Migration repository tree example:

```
1.0/
    1.0-0-version-dml.sql
    1.0-feature_a-010-ddl.sql
    1.0-feature_a-020-dml.sql
1.1/
    1.1-0-version-dml.sql
2.0/
    2.0-0-version-dml.sql
2.1/
    2.1-0-version-dml.sql
fixtures/
    fixtures_1.0.sql
    fixtures_1.1.sql
    fixtures_2.0.sql
schemas/
    schema_1.0.sql
    schema_1.1.sql
    schema_2.0.sql
```

See also some examples in tests/test_data/sql folder (used for unit tests), or in tests/north_project/sql folder (used for realistic tests).

The migrations are alphabetical ordered.

Only files which name ends with ddl.sql and dml.sql are run.

3.1.1 Currect version detector

django-north needs to know the current version, to init or upgrade the database schema. Because it depends on your contract with the DBA team, it is possible to customize the current version detector.

django-north provides two detectors:

- django_north.management.migrations.get_current_version_from_table
- django_north.management.migrations.get_current_version_from_comment

But you can also write your own detector.

Just set the NORTH_CURRENT_VERSION_DETECTOR setting to use the chosen one.

It is possible to write a detector which just queries the django_migration table, but we decided to implement solutions that do not depend on migrations themselves: the django_migration table is filled by the django-north tool, which is not used in production environments.

It can be usefull to have this information in production environments: you will be able to check that you can deploy a new code version.

From Table

The default detector.

The first schema has to create a table like:

```
CREATE TABLE sql_version (
    version_num text UNIQUE NOT NULL
);
```

Init the version in the corresponding fixtures file (dml):

```
INSERT INTO sql_version(version_num) VALUES ('1.0');
```

And the version upgrade in the first migration of each version (a dml file):

```
INSERT INTO sql_version(version_num) VALUES ('2.0');
```

From Comment

For this detector you need to have a `django_site` table.

Init the version in the schema (ddl):

```
COMMENT ON TABLE django_site IS 'version 1.0';
```

And the version upgrade in the first migration of each version (a dml file):

```
COMMENT ON TABLE django_site IS 'version 2.0';
```

3.2 Manual migrations

A "manual" migration file is a dml migration which should be run more than once.

For example, if you have a big table with a lot of data, and a data migration to do, you probably would like to run the migration by chunks.

Manual migration files can stored in the "manual" subdirectory of a version directory:

```
1.0/
    manual/
        1.0-0-version-dml.sql
    1.0-feature_a-010-ddl.sql
    1.0-feature_a-020-dml.sql
```

Else, a migration file can be considered as a manual migration file if:

- the end of the migration file name is `dml.sql`
- and it contains a meta instruction `--meta-psql:`

3.2.1 Meta instructions

do-until-0

Example:

```
BEGIN;


-- example of a manual migration


--meta-psql:do-until-0

with to_update as (
    SELECT
        id
    FROM north_app_book
    WHERE num_pages = 0
    LIMIT 5000
)
UPDATE north_app_book SET num_pages = 42 WHERE id IN (
    SELECT id FROM to_update
);

--meta-psql:done


COMMIT;
```

3.3 Available Commands

3.3.1 migrate

```
$ ./tests_manage.py migrate
```

Create a DB from scratch and migrate it to the version defined in the `NORTH_TARGET_VERSION` setting, or update an existing DB to migrate it to the correct version.

This command knows which migrations are already applied, which migrations should be applied.

This command can only go forward: no possible revert like with south or django migrations. But as the migrations written by the DBA team are blue/green, that is not a problem !

This command has no effects if the `NORTH_MANAGE_DB` setting is disabled.

3.3.2 showfixtures

```
$ ./tests_manage.py showfixtures
```

List missing fixtures, and print SQL instructions to create them (ask your DBA team to add a dml migration for that).

"Fixtures" designates here datas which are automatically created by django on `post_migrate` signal, and required for the project.

Basically:

- content types (`django.contrib.contenttypes`)
- permissions (`django.contrib.auth`)

The site id 1 (`SITE_ID` setting) is not checked by this command.

Note: When you add a Model, you have to run this command twice to get: 1/ the new content type 2/ when the content type exists, the new permissions

---

3.3.3 showmigrations

```
$ ./tests_manage.py showmigrations
```

List available migrations, and indicate if they where applied or not.

This command has no effects if the `NORTH_MANAGE_DB` setting is disabled.

3.4 Changed Commands

3.4.1 sqlall

Django >= 1.9: the command is backported.

```
$ ./tests_manage.py sqlall <app>
```

Usefull to print the CREATE TABLE and CREATE INDEX SQL statements for the init of a DB schema, for an external app with a migration folder (as `django.contrib.auth` app for example).

3.4.2 flush

```
$ ./tests_manage.py flush
```

Did a truncate on all tables, where the original command did it only on tables defined in the django models.

Reload the SQL fixtures, and reset the ContentType cache.

This command is essential for the tests, especially for TransactionTestCase tests.

This command has no effects if the `NORTH_MANAGE_DB` setting is disabled.

3.4.3 runserver

```
$ ./tests_manage.py runserver
```

Display a warning if some migrations are not applied.

3.5 Disabled Commands

These commands are disabled whatever the value of the `NORTH_MANAGE_DB` setting:

- `makemigrations`
- `sqlmigrate`
- `squashmigrations`

3.6 Tips

3.6.1 Generate Schema Files

At the end of a SQL release, just do a sqldump (`pg_dump -s` for posgtres for example).

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/peopledoc/django-north/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

4.1.4 Write Documentation

Django North could always use more documentation, whether as part of the official Django North docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/peopledoc/django-north/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *django-north* for local development.

1. Fork the *django-north* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-north.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-north
$ cd django-north/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 django_north tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7. Check https://travis-ci.org/peopledoc/django-north/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 5

Credits

5.1 Development Lead

- Lauréline Guérin / @zebuline <laureline.guerin /at/ people-doc.com>

5.2 Contributors (by alphabetic order)

- @ewjoachim

CHAPTER 6

History

6.1 0.2.7 (unreleased)

- Nothing changed yet.

6.2 0.2.6 (2019-10-25)

- Add support for Django 2.2
- *NORTH_AFTER_SCHEMA_FILES* and *NORTH_BEFORE_SCHEMA_FILES* can now accept glob string.

6.3 0.2.5 (2019-01-22)

- Add support for Django 2.1 & Python 3.7
- Add setting *NORTH_AFTER_SCHEMA_FILES* for schema files after the main schema.
- Adding setting *NORTH_BEFORE_SCHEMA_FILES*, to replace *NORTH_ADDITIONAL_SCHEMA_FILES*.
- Deprecate setting *NORTH_ADDITIONAL_SCHEMA_FILES*.

6.4 0.2.4 (2018-09-12)

- Use *–database* option to determine which database to use in migrate command (#35)

6.5 0.2.3 (2018-06-15)

- Add support for Django 2.0 (#31)

- Add a "DISCARD ALL" command run at the end of each script. It adds a new settings variable: `NORTH_DISCARD_ALL` (#33)

6.6 0.2.2 (2018-02-01)

- Flush command: do not flush migration tables.

6.7 0.2.1 (2018-01-29)

- Add *VACUUM* to *NORTH_NON_TRANSACTIONAL_KEYWORDS* default settings.
- Add a setting *NORTH_SCHEMA_VERSION* to force the schema to be used to init a DB.

6.8 0.2.0 (2017-10-16)

- Backport the *sqlall* command.
- Sanitize sql statements for SimpleBlock.

6.9 0.1.8 (2017-09-20)

- Detect manual files if not stored in the 'manual' dir.
- Fix unicode error with SimpleBlock

6.10 0.1.7 (2017-09-06)

- Fix *get_applied_versions* result ordering.

6.11 0.1.6 (2017-09-05)

- Add tests for Django 1.11.

6.12 0.1.5 (2017-05-24)

- Fix showfixtures command for Django 1.10.

6.13 0.1.4 (2017-05-10)

- Do not fail if fixtures do not exist. Use the closest fixtures for DB init and flush command.
- Add support of python3.

6.14 0.1.3 (2017-04-18)

- Use a Block if the sql file contains a 'ALTER TYPE' instruction Add a setting to customize the files to run in a Block.

6.15 0.1.2 (2017-04-13)

- Use a Block if the sql file contains a CONCURRENTLY instruction.

6.16 0.1.1 (2017-04-11)

- Add the possibility to configure the current version detector.

6.17 0.1.0 (2017-03-28)

- First release on PyPI.