nashvegas Documentation

Release 0.7

Patrick Altman

Sep 27, 2017

Contents

The purpose of this app is to enable a plug and play method for managing database changes.

Database migrations is a large topic with a lot of different approaches. This approach worked well for my needs and maybe it will for you as well.

CHAPTER 1

Installation

- pip install nashvegas
- Add the application to your INSTALLED_APPS list in your settings.py file.

CHAPTER 2

Settings

You can set the `NASHVEGAS_MIGRATIONS_DIRECTORY` to whatever absolute path you are using to store your migrations. It defaults to `migrations/` at the same level as your `settings.py`.

CHAPTER 3

Usage

nashvegas ships with two management commands, `upgradedb` and `comparedb`. The first, `upgradedb`, will manage the creation, listing, and execution of individual migrations. The second, `comparedb`, is currently an experimental command that attempts to help you discover missing migrations.

- Execute the command line:

  $ ./manage.py upgradedb –create|–list|–execute $ ./manage.py comparedb

CHAPTER 4

Options for upgradedb

- `--create` - Compares database with current models in apps that are installed and outputs the sql for them so that you can easily pipe the contents to a migration.
- `--list` - Lists all the scripts that will need to be executed.
- `--execute` - Executes all the scripts that need to be executed.
- `--seed` - Populates Migration model with scripts that have already been applied to your database and effectively want to skip execution. Provide a migration id to stop at. For instance, running *./manage.py upgradedb –seed 005* will skip migrations 000 to 005 but not 006.

CHAPTER 5

Configuration for comparedb

The *comparedb* command is available only for Postgres. It executes a few raw postgres shell commands which you might need to customize to add user credentials, encoding or specify database templates. This can be done through the *NASHVEGAS* dictionnary in your setting:

```
NASHVEGAS = {
    "createdb": "createdb -U postgres -T template0 -E UTF8",
    "dropdb": "dropdb -U postgres",
    "pg_dump": "pg_dump -U postgres",
}
```

By default, nashvegas executes raw *createdb*, *dropdb* or *pg_dump* commands.

CHAPTER 6

Conventions

Part of the simplicity of this solution is based on the naming conventions of the sql scripts. They should be named in a manner that enforces order. Some examples include:

```
0001_short_comment_about_migration.sql
0001.sql
```

The model, `nashvegas.Migration` will get synced into your database if it doesn't exist when you go to execute any of the `upgradedb` commands. In this model the scripts that have been executed will be recorded, effectively versioning your database.

In addition to sql scripts, `--execute` will also execute python scripts that are in the directory. This are run in filename order interleaved with the sql scripts. For example:

```
0001.sql
0002.py
0003.sql
```

The Python script will be executed 2nd between `0000.sql` and `0003.sql`. The script will only be executed if the module contains a callable named `migrate`. It is a good idea to put all your executing code within a class or series of functions or within a singe `migrate()` function so as to avoid code executing upon import.

For example, your script might light like this if you need to update all your product codes on next release:

```
from store.models import Product

def migrate():
    for product in Product.objects.all():
        product.code = "NEW-%s" % product.code
        product.save()
```

Indices and tables

- genindex

- modindex
- search