django-cities-light Documentation

Release 3.2.0

James Pic

May 09, 2016

Contents

This add-on provides models and commands to import country, region/state, and city data in your database.

The data is pulled from GeoNames and contains cities, regions/states and countries.

Spatial query support is not required by this application.

This application is very simple and is useful if you want to make a simple address book for example. If you intend to build a fully featured spatial database, you should use django-cities.

Requirements:

- Python 2.7 or 3.3,
- Django >= 1.7
- MySQL or PostgreSQL or SQLite.

Yes, for some reason, code that used to work on MySQL (not without pain xD) does not work anymore. So we're now using django.db.transaction.atomic which comes from Django 1.6 just to support MySQL quacks.

CHAPTER 1

Upgrade

See CHANGELOG.

CHAPTER 2

Installation

Install django-cities-light:

```
pip install django-cities-light
```

Or the development version:

```
pip install -e git+git@github.com:yourlabs/django-cities-light.git#egg=cities_light
```

Add *cities_light* to your *INSTALLED_APPS*.

Configure filters to exclude data you don't want, ie.:

```
CITIES_LIGHT_TRANSLATION_LANGUAGES = ['fr', 'en']
CITIES_LIGHT_INCLUDE_COUNTRIES = ['FR']
CITIES_LIGHT_INCLUDE_CITY_TYPES = ['PPL', 'PPLA', 'PPLA2', 'PPLA3', 'PPLA4', 'PPLC', 'PPLF', 'PPLG',
```

Now, run migrations, it will only create tables for models that are not disabled:

```
./manage.py migrate
```

CHAPTER 3

Data update

Finally, populate your database with command:

```
./manage.py cities_light
```

This command is well documented, consult the help with:

```
./manage.py help cities_light
```

CHAPTER 4

Resources

You could subscribe to the mailing list ask questions or just be informed of package updates.

- Mailing list graciously hosted by Google
- Git graciously hosted by GitHub,
- Documentation graciously hosted by RTFD,
- Package graciously hosted by PyPi,
- Continuous integration graciously hosted by Travis-ci
- **Online paid support** provided via HackHands,

Contents:

4.1 Populating the database

4.1.1 Data install or update

Populate your database with command:

```
./manage.py cities_light
```

By default, this command attempts to do the least work possible, update what is necessary only. If you want to disable all these optimisations/skips, use –force-all.

This command is well documented, consult the help with:

```
./manage.py help cities_light
```

4.1.2 Signals

Signals for this application.

`cities_light.signals.`**`city_items_pre_import`**

Emited by city_import() in the cities_light command for each row parsed in the data file. If a signal reciever raises InvalidItems then it will be skipped.

An example is worth 1000 words: if you want to import only cities from France, USA and Belgium you could do as such:

```
import cities_light

def filter_city_import(sender, items, **kwargs):
    if items[8] not in ('FR', 'US', 'BE'):
        raise cities_light.InvalidItems()

cities_light.signals.city_items_pre_import.connect(filter_city_import)
```

Note: this signal gets a list rather than a City instance for performance reasons.

`cities_light.signals.`**`region_items_pre_import`**
Same as *city_items_pre_import*.

`cities_light.signals.`**`country_items_pre_import`**
Same as *region_items_pre_import* and *cities_light.signals.city_items_pre_import*.

`cities_light.signals.`**`city_items_post_import`**
Emited by city_import() in the cities_light command for each row parsed in the data file, right before saving City object. Along with City instance it pass items with geonames data. Will be useful, if you define custom cities models with `settings.CITIES_LIGHT_APP_NAME`.

Example:

```
import cities_light

def process_city_import(sender, instance, items, **kwargs):
    instance.timezone = items[17]

cities_light.signals.city_items_post_import.connect(process_city_import)
```

`cities_light.signals.`**`region_items_post_import`**
Same as *city_items_post_import*.

`cities_light.signals.`**`country_items_post_import`**
Same as *region_items_post_import* and *cities_light.signals.city_items_post_import*.

exception `cities_light.exceptions.`**`CitiesLightException`**
Base exception class for this app's exceptions.

exception `cities_light.exceptions.`**`InvalidItems`**
The cities_light command will skip item if a city_items_pre_import signal reciever raises this exception.

exception `cities_light.exceptions.`**`SourceFileDoesNotExist`**(*source*)
A source file could not be found.

4.1.3 Configure logging

This command is made to be compatible with background usage like from cron, to keep the database fresh. So it doesn't do direct output. To get output from this command, simply configure a handler and formatter for *cities_light* logger. For example:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
    },
    'handlers': {
        'console':{
            'level':'DEBUG',
            'class':'logging.StreamHandler',
            'formatter': 'simple'
        },
    },
    'loggers': {
        'cities_light': {
            'handlers':['console'],
            'propagate': True,
            'level':'DEBUG',
        },
        # also use this one to see SQL queries
        'django': {
            'handlers':['console'],
            'propagate': True,
            'level':'DEBUG',
        },
    }
}
```

4.2 Simple django app

4.2.1 Settings

Settings for this application. The most important is TRANSLATION_LANGUAGES because it's probably project specific.

`cities_light.settings.`**`TRANSLATION_LANGUAGES`**

List of language codes. It is used to generate the alternate_names property of cities_light models. You want to keep it as small as possible. By default, it includes the most popular languages according to wikipedia, which use a rather ascii-compatible alphabet. It also contains 'abbr' which stands for 'abbreviation', you might want to include this one as well.

See:

- http://download.geonames.org/export/dump/iso-languagecodes.txt

Example:

```
CITIES_LIGHT_TRANSLATION_LANGUAGES = ['es', 'en', 'fr', 'abbr']
```

`cities_light.settings.`**`INCLUDE_COUNTRIES`**

List of country codes to include. It's None by default which lets all countries in the database. But if you only wanted French and Belgium countries/regions/cities, you could set it as such:

```
CITIES_LIGHT_INCLUDE_COUNTRIES = ['FR', 'BE']
```

`cities_light.settings.`**`INCLUDE_CITY_TYPES`**

List of city feature codes to include. They are described at http://www.geonames.org/export/codes.html, section "P city, village".

CITIES_LIGHT_INCLUDE_CITY_TYPES = ['PPL', 'PPLA', 'PPLA2', 'PPLA3', 'PPLA4', 'PPLC', 'PPLF', 'PPLG', 'PPLL', 'PPLR', 'PPLS', 'STLMT',

]

`cities_light.settings.`**`COUNTRY_SOURCES`**
A list of urls to download country info from. Default is countryInfo.txt from geonames download server. Overridable in `settings.CITIES_LIGHT_COUNTRY_SOURCES`.

`cities_light.settings.`**`REGION_SOURCES`**
A list of urls to download region info from. Default is admin1CodesASCII.txt from geonames download server. Overridable in `settings.CITIES_LIGHT_REGION_SOURCES`.

`cities_light.settings.`**`CITY_SOURCES`**
A list of urls to download city info from. Default is cities15000.zip from geonames download server. Overridable in `settings.CITIES_LIGHT_CITY_SOURCES`.

`cities_light.settings.`**`TRANSLATION_SOURCES`**
A list of urls to download alternate names info from. Default is alternateNames.zip from geonames download server. Overridable in `settings.CITIES_LIGHT_TRANSLATION_SOURCES`.

`cities_light.settings.`**`SOURCES`**
A list with all sources, auto-generated.

`cities_light.settings.`**`DATA_DIR`**
Absolute path to download and extract data into. Default is cities_light/data. Overridable in `settings.CITIES_LIGHT_DATA_DIR`

`cities_light.settings.`**`INDEX_SEARCH_NAMES`**
If your database engine for cities_light supports indexing TextFields (ie. it is **not** MySQL), then this should be set to True. You might have to override this setting with `settings.CITIES_LIGHT_INDEX_SEARCH_NAMES` if using several databases for your project.

`cities_light.settings.`**`CITIES_LIGHT_APP_NAME`**
Modify it only if you want to define your custom cities models, that are inherited from abstract models of this package. It must be equal to app name, where custom models are defined. For example, if they are in geo/models.py, then set `settings.CITIES_LIGHT_APP_NAME = 'geo'`. Note: you can't define one custom model, you have to define all of cities_light models, even if you want to modify only one.

class `cities_light.settings.`**`ICountry`**
Country field indexes in geonames.

class `cities_light.settings.`**`IRegion`**
Region field indexes in geonames.

class `cities_light.settings.`**`ICity`**
City field indexes in geonames. Description of fields: http://download.geonames.org/export/dump/readme.txt

class `cities_light.settings.`**`IAlternate`**
Alternate names field indexes in geonames. Description of fields: http://download.geonames.org/export/dump/readme.txt

4.2.2 Models

See source for details.

4.2.3 Admin

See source for details.

4.3 cities_light.contrib

4.3.1 For django-ajax-selects

4.3.2 For djangorestframework

The contrib contains support for v1, v2 and v3 of django restframework.

Django REST framework 3

This contrib package defines list and detail endpoints for City, Region and Country. If rest_framework (v3) is installed, all you have to do is add this url include:

```
url(r'^cities_light/api/', include('cities_light.contrib.restframework3')),
```

This will configure six endpoints:

```
^cities/$ [name='cities-light-api-city-list']
^cities/(?P<pk>[^/]+)/$ [name='cities-light-api-city-detail']
^countries/$ [name='cities-light-api-country-list']
^countries/(?P<pk>[^/]+)/$ [name='cities-light-api-country-detail']
^regions/$ [name='cities-light-api-region-list']
^regions/(?P<pk>[^/]+)/$ [name='cities-light-api-region-detail']
```

All list endpoints support search with a query parameter q:: /cities/?q=london

For Region and Country endpoints, the search will be within name_ascii field while for City it will search in search_names field. HyperlinkedModelSerializer is used for these models and therefore every response object contains url to self field and urls for related models. You can configure pagination using the standard rest_framework pagination settings in your project settings.py.

4.3.3 Ideas for contributions

- templatetag to render a city's map using some external service
- flag images, maybe with django-countryflags
- currencies
- generate po files when parsing alternate names

CHAPTER 5

FAQ

5.1 Recommended RDBMS

The recommended RDBMS is PostgreSQL, it's faster, safer, saner, more robust and simpler than MySQL.

You can see on travis that build jobs with MySQL take twice as long as build jobs on PostgreSQL and SQLite.

5.2 MySQL errors with special characters, how to fix it ?

The `cities_light` command is continuously tested on travis-ci on all supported databases: if it works there then it should work for you.

If you're new to development in general, you might not be familiar with the concept of encodings and collations. Unless you have a good reason, you **must** have utf-8 database tables. See MySQL documentation for details.

We're pointing to MySQL documentations because PostgreSQL users probably know what UTF-8 is and won't have any problem with that.

5.3 Some data fail to import, how to skip them ?

GeoNames is not perfect and there might be some edge cases from time to time. We want the `cities_light` management command to work for everybody so you should open an issue in GitHub if you get a crash from that command.

However, we don't want you to be blocked, so keep in mind that you can use *Signals* like *`cities_light.city_items_pre_import`*, *`cities_light.region_items_pre_import`*, *`cities_light.country_items_pre_import`*, to skip or fix items before they get inserted in the database by the normal process.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index