sqlalchemyseed

jedymatt

Jan 14, 2022

CONTENTS

SQLAlchemy seeder that supports nested relationships with an easy to read text files.

Project Links: Github | PyPI

CHAPTER

ONE

INTRODUCTION

sqlalchemyseed is a SQLAlchemy seeder that supports nested relationships with an easy to read text files.

Supported file types :

- json
- yaml
- csv

1.1 Installation

Default installation

```
pip install sqlalchemyseed
```

When using yaml to load entities from yaml files, execute this command to install necessary dependencies

```
pip install sqlalchemyseed[yaml]
```

1.2 Dependencies

Required dependencies:

- SQAlchemy>=1.4.0

Optional dependencies:

- yaml
  - PyYAML>=5.4.0

1.3 Quickstart

Here's a simple snippet to get started from `main.py` file.

```
from sqlalchemyseed import load_entities_from_json
from sqlalchemyseed import Seeder
from db import session

# load entities
entities = load_entities_from_json('data.json')

# Initializing Seeder
seeder = Seeder(session)

# Seeding
seeder.seed(entities)

# Committing
session.commit()  # or seeder.session.commit()
```

And the `data.json` file.

```
{
    "model": "models.Person",
    "data": [
        {
            "name": "John March",
            "age": 23
        },
        {
            "name": "Juan Dela Cruz",
            "age": 21
        }
    ]
}
```

CHAPTER

TWO

SEEDING

2.1 Seeder vs. HybridSeeder

Features & Options	Seeder	HybridSeeder
Support `model` and `data` keys	✓	✓
Support `model` and `filter` keys		✓
Optional argument `add_to_session=False` in the `seed` method	✓	

2.2 When to use HybridSeeder and 'filter' key field?

Assuming that `Child(age=5)` exists in the database or session, then we should use `filter` instead of `data` key.

The values from `filter` will query from the database or session, and get the result then assign it to the `Parent.child`

```
from sqlalchemyseed import HybridSeeder
from db import session

data = {
    "model": "models.Parent",
    "data": {
        "!child": { # '!' is the reference prefix
            "model": "models.Child",
            "filter": {
                "age": 5
            }
        }
    }
}

# When seeding instances that has 'filter' key,
# then use HybridSeeder, otherwise use Seeder.
seeder = HybridSeeder(session, ref_prefix='!')
seeder.seed(data)

session.commit()  # or seeder.sesssion.commit()
```

Note: `filter` key is dependent to HybridSeeder in order to perform correctly.

CHAPTER

THREE

REFERENCING RELATIONSHIPS

To add reference attribute, add prefix to the attribute to differentiate reference attribute from normal ones.

```
{
    "model": "models.Employee",
    "data": {
        "name": "John Smith",
        "!company": {
            "model": "models.Company",
            "data": {
                "name": "MyCompany"
            }
        }
    }
}
```

Base on the example above, **name** is a normal attribute and **!company** is a reference attribute which translates to `Employee.name` and `Employee.company`, respectively.

Note: The default reference prefix is `!` and can be customized.

3.1 Customizing reference prefix

If you want `@` as prefix, you can just specify it to what seeder you use by assigning value of `Seeder.ref_prefix` or `HybridSeeeder.ref_prefix`. Default value is `!`

```
seeder = Seeder(session, ref_prefix='@')
# or
seeder = Seeder(session)
seeder.ref_prefix = '@'
```

3.2 Types of reference attributes

Reference attribute types:

- foreign key attribute
- relationship attribute

You can reference a foreign key and relationship attribute in the same way. For example:

```
from sqlalchemyseed import HybridSeeder
from db import session

instance = {
        'model': 'tests.models.Employee',
        'data': [
            {
                'name': 'John Smith',
                '!company_id': {  # this is the foreign key attribute
                    'model': 'tests.models.Company',
                    'filter': {
                        'name': 'MyCompany'
                    }
                }
            },
            {
                'name': 'Juan Dela Cruz',
                '!company': { # this is the relationship attribute
                    'model': 'tests.models.Company',
                    'filter': {
                        'name': 'MyCompany'
                    }
                }
        ]
    }

seeder = HybridSeeder(session)
seeder.seed(instance)
seeder.session.commit()
```

Note: `model` can be removed if the attribute is a reference attribute like this:

```
{
    "model": "models.Employee",
    "data": {
        "name": "Juan Dela Cruz",
        "!company": {
            "data": {
                "name": "Juan's Company"
            }
        }
    }
}
```

Notice above that `model` is removed in `!company`.

CHAPTER

FOUR

EXAMPLES

4.1 json

```
{
     "model": "models.Person",
     "data": [
         {
             "name": "John March",
             "age": 23
         },
         {
             "name": "Juan Dela Cruz",
             "age": 21
         }
     ]
 }
```

4.2 yaml

```
model: models.Person
data:
    - name: John March
    age: 23
    - name: Juan Dela Cruz
    age: 21
```

4.3 csv

In line one, name and age, are attributes of a model that will be specified when loading the file.

```
name, age
John March, 23
Juan Dela Cruz, 21
```

To load a csv file

```
# second argument, model, accepts class
load_entities_from_csv("people.csv", models.Person)
# or string
load_entities_from_csv("people.csv", "models.Person")
```

Note: csv does not support referencing relationships.

4.4 No Relationship

```
[
    {
        "model": "models.Person",
        "data": {
            "name": "You",
            "age": 18
        }
    },
    {
        "model": "models.Person",
        "data": [
            {
                "name": "You",
                "age": 18
            },
            {
                "name": "Still You But Older",
                "age": 40
            }
        ]
    }
]
```

4.5 One to One Relationship

```
[
    {
        "model": "models.Person",
        "data": {
            "name": "John",
            "age": 18,
            "!job": {
                "model": "models.Job",
                "data": {
                    "job_name": "Programmer",
                }
            }
```

(continues on next page)

(continued from previous page)

```
        }
    },
    {
        "model": "models.Person",
        "data": {
            "name": "Jeniffer",
            "age": 18,
            "!job": {
                "model": "models.Job",
                "filter": {
                    "job_name": "Programmer",
                }
            }
        }
    }
]
```

4.6 One to Many Relationship

```
[
    {
        "model": "models.Person",
        "data": {
            "name": "John",
            "age": 18,
            "!items": [
                {
                    "model": "models.Item",
                    "data": {
                        "name": "Pencil"
                    }
                },
                {
                    "model": "models.Item",
                    "data": {
                        "name": "Eraser"
                    }
                }
            ]
        }
    }
]
```

Nested Relationships

```
{
    "model": "models.Parent",
    "data": {
        "name": "John Smith",
        "!children": [
```

(continues on next page)

(continued from previous page)

```
            {
                "model": "models.Child",
                "data": {
                    "name": "Mark Smith",
                    "!children": [
                        {
                            "model": "models.GrandChild",
                            "data": {
                                "name": "Alice Smith"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

CHAPTER

FIVE

API REFERENCE

5.1 Seeders

class sqlalchemyseed.**Seeder**(*session: Optional[sqlalchemy.orm.session.Session] = None*, *ref_prefix='!'*)

get_model_class(*entity*, *parent: sqlalchemyseed.seeder.Entity*)

property instances

seed(*entities*, *add_to_session=True*)

class sqlalchemyseed.**HybridSeeder**(*session: sqlalchemy.orm.session.Session*, *ref_prefix: str = '!'*)

get_model_class(*entity*, *parent: sqlalchemyseed.seeder.Entity*)

property instances

seed(*entities*)

5.2 Loaders

sqlalchemyseed.**load_entities_from_json**(*json_filepath*)

sqlalchemyseed.**load_entities_from_yaml**(*yaml_filepath*)

sqlalchemyseed.**load_entities_from_csv**(*csv_filepath: str*, *model*) → dict

Load entities from csv file

Parameters

- **csv_filepath** – string csv file path
- **model** – either str or class

Returns dict of entities

5.3 Validators

sqlalchemyseed.validator.**validate**(*entities*, *ref_prefix='!'*)

sqlalchemyseed.validator.**hybrid_validate**(*entities*, *ref_prefix='!'*)

5.4 Exceptions

exception sqlalchemyseed.errors.**ClassNotFoundError**
Raised when the class is not found

exception sqlalchemyseed.errors.**EmptyDataError**
Raised when data is empty

exception sqlalchemyseed.errors.**InvalidKeyError**
Raised when an invalid key is invoked

exception sqlalchemyseed.errors.**InvalidTypeError**
Raised when a type of data is not accepted

exception sqlalchemyseed.errors.**MaxLengthExceededError**
Raised when maximum length of data exceeded

exception sqlalchemyseed.errors.**MissingKeyError**
Raised when a required key is missing

exception sqlalchemyseed.errors.**NotInModuleError**
Raised when a value is not found in module

exception sqlalchemyseed.errors.**ParseError**
Raised when parsing string fails

exception sqlalchemyseed.errors.**UnsupportedClassError**
Raised when an unsupported class is invoked

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX