
sqlalchemyseed

jedymatt

Mar 12, 2023

CONTENTS:

- 1 Introduction 3**
 - 1.1 Installation 3
 - 1.2 Dependencies 3
 - 1.3 Quickstart 3
- 2 Seeding 5**
 - 2.1 Seeder vs. HybridSeeder 5
 - 2.2 When to use HybridSeeder and ‘filter’ key field? 5
- 3 Referencing Relationships 7**
 - 3.1 Customizing reference prefix 7
 - 3.2 Types of reference attributes 8
- 4 Examples 11**
 - 4.1 json 11
 - 4.2 yaml 11
 - 4.3 csv 11
 - 4.4 No Relationship 12
 - 4.5 One to One Relationship 12
 - 4.6 One to Many Relationship 13
- 5 API Reference 15**
 - 5.1 sqlalchemyseed 15
- 6 Indices and tables 21**
- Python Module Index 23**
- Index 25**

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

Project Links: [Github](#) | [PyPI](#)

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:**
 - SQLAlchemy>=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
```

(continues on next page)

(continued from previous page)

```
# 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
    }
  ]
}
```


2.1 Seeder vs. HybridSeeder

Features & Options	Seeder	HybridSeeder
Support model and data keys	✓	✓
Support model and filter keys		✓
Optional argument <code>add_to_session=False</code> 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.session.commit()
```

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

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 `HybridSeeder.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`.

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"
        }
      }
    ]
  }
}
```

API REFERENCE

This page contains auto-generated API reference documentation¹.

5.1 sqlalchemy.seed

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

5.1.1 Submodules

`sqlalchemy.seed.attribute`

attribute module containing helper functions for instrumented attribute.

Module Contents

`sqlalchemy.seed.attribute.attr_is_column(instrumented_attr: sqlalchemy.orm.attributes.InstrumentedAttribute)`

Check if instrumented attribute property is a ColumnProperty

`sqlalchemy.seed.attribute.attr_is_relationship(instrumented_attr: sqlalchemy.orm.attributes.InstrumentedAttribute)`

Check if instrumented attribute property is a RelationshipProperty

`sqlalchemy.seed.attribute.foreign_key_column(instrumented_attr: sqlalchemy.orm.attributes.InstrumentedAttribute)`

Returns the table name of the first foreignkey.

`sqlalchemy.seed.attribute.instrumented_attribute(class_or_instance, key: str)`

Returns instrumented attribute from the class or instance.

`sqlalchemy.seed.attribute.referenced_class(instrumented_attr: sqlalchemy.orm.attributes.InstrumentedAttribute)`

Returns class that the attribute is referenced to.

`sqlalchemy.seed.attribute.set_instance_attribute(instance, key, value)`

Set attribute value of instance

¹ Created with sphinx-autoapi

sqlalchemyseed.constants

Module Contents

```
sqlalchemyseed.constants.DATA_KEY = 'data'
sqlalchemyseed.constants.FILTER_KEY = 'filter'
sqlalchemyseed.constants.MODEL_KEY = 'model'
sqlalchemyseed.constants.SOURCE_KEYS
```

sqlalchemyseed.dynamic_seeder

Module Contents

```
class sqlalchemyseed.dynamic_seeder.DynamicSeeder
    DynamicSeeder class
```

sqlalchemyseed.errors

Module Contents

```
exception sqlalchemyseed.errors.ClassNotFoundError
    Bases: Exception
    Raised when the class is not found

exception sqlalchemyseed.errors.EmptyDataError
    Bases: Exception
    Raised when data is empty

exception sqlalchemyseed.errors.InvalidKeyError
    Bases: Exception
    Raised when an invalid key is invoked

exception sqlalchemyseed.errors.InvalidModelPath
    Bases: Exception
    Raised when an invalid model path is invoked

exception sqlalchemyseed.errors.InvalidTypeError
    Bases: Exception
    Raised when a type of data is not accepted

exception sqlalchemyseed.errors.MaxLengthExceededError
    Bases: Exception
    Raised when maximum length of data exceeded
```

exception sqlalchemyseed.errors.**MissingKeyError**

Bases: Exception

Raised when a required key is missing

exception sqlalchemyseed.errors.**NotInModuleError**

Bases: Exception

Raised when a value is not found in module

exception sqlalchemyseed.errors.**ParseError**

Bases: Exception

Raised when parsing string fails

exception sqlalchemyseed.errors.**UnsupportedClassError**

Bases: Exception

Raised when an unsupported class is invoked

sqlalchemyseed.json

Module Contents

class sqlalchemyseed.json.**JsonWalker**(*json: list | dict = None*)

JsonWalker class

property **current_key**: int | str

Returns the key of the current json

property **json**

Returns current json

property **json_is_dict**

Returns true if current json is dict

property **json_is_list**

Returns true if current json is list

backward()

Revert current json to its parent. Returns reverted current json

exec_func_iter(*func: Callable*)

Executes function when iterating

find_from_current(*keys: List[int | str]*)

Find item from current json that correlates list of keys

find_from_root(*keys: List[int | str]*)

Find item from the root json that correlates list of keys

forward(*keys: List[int | str]*)

Move and replace current json forward. Returns current json.

iter_as_dict_items()

Iterates current as dict. Yields key and value.

Raises TypeError if current json is not dict

iter_as_list()

Iterates current as list. Yields index and value.

Raises TypeError if current json is not list

keys()

Returns list of keys either str or int

reset(*root=None*)

Resets to initial state. If root argument is supplied, self.root will be replaced.

`sqlalchemyseed.json.sort_json(json: list | dict, reverse=False)`

Sort json function

sqlalchemyseed.loader

Text file loader module

Module Contents

`sqlalchemyseed.loader.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

`sqlalchemyseed.loader.load_entities_from_json(json_filepath) → dict`

Get entities from json

`sqlalchemyseed.loader.load_entities_from_yaml(yaml_filepath)`

Get entities from yaml

sqlalchemyseed.seeder

Seeder module

Module Contents

class `sqlalchemyseed.seeder.AbstractSeeder`

Bases: `abc.ABC`

AbstractSeeder class

abstract property instances

Seeded instances

abstract seed(*entities*)

Seed data

class sqlalchemyseed.seeder.**DynamicSeeder**

DynamicSeeder class

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

Bases: [AbstractSeeder](#)

HybridSeeder class. Accepts 'filter' key for referencing children.

property instances

Seeded instances

get_model_class(*entity, parent: InstanceAttributeTuple*)

seed(*entities*)

Seed data

class sqlalchemyseed.seeder.**InstanceAttributeTuple**

Bases: [NamedTuple](#)

Instance and attribute name tuple

attr_name: str

instance: object

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

Basic Seeder class

property instances: tuple

Returns instances of the seeded entities

seed(*entities: list | dict, add_to_session=True*)

Seed method

sqlalchemyseed.seeder.**filter_kwargs**(*kwargs: dict, class_, ref_prefix*)

Filters kwargs

sqlalchemyseed.util

Utility functions

Module Contents

sqlalchemyseed.util.**find_item**(*json: Iterable, keys: list*)

Finds item of json from keys

sqlalchemyseed.util.**generate_repr**(*instance: object*) → str

Generate repr of object instance

sqlalchemyseed.util.**get_model_class**(*path: str*)

Get sqlalchemy model class from path

sqlalchemyseed.util.**is_model**(*class_*)

Check if class is a sqlalchemy model

`sqlalchemyseed.util.is_supported_class(class_)`
Check if it is a class and supports sqlalchemy

`sqlalchemyseed.util.iter_kwargs_with_prefix(kwargs: dict, prefix: str)`
Iterate kwargs(dict) that has the specified prefix.

`sqlalchemyseed.util.iter_non_ref_kwargs(kwargs: dict, ref_prefix: str)`
Iterate kwargs, skipping item with name prefix or references

`sqlalchemyseed.util.iter_ref_kwargs(kwargs: dict, ref_prefix: str)`
Iterate kwargs with name prefix or references

`sqlalchemyseed.util.iterate_json(json: dict, key_prefix: str)`
Iterate through json that has matching key prefix

`sqlalchemyseed.util.iterate_json_no_prefix(json: dict, key_prefix: str)`
Iterate through json that has no matching key prefix

sqlalchemyseed.validator

Validator module.

Module Contents

`class sqlalchemyseed.validator.Key(name: str, type_)`

- `classmethod data()`
- `classmethod filter()`
- `is_valid_type(entity)`
- `classmethod model()`

`class sqlalchemyseed.validator.SchemaValidator(source_keys, ref_prefix)`

- `check_attributes(source_data: dict)`
- `validate(entities)`

`sqlalchemyseed.validator.check_data_type(item, source_key: Key)`

`sqlalchemyseed.validator.check_max_length(entity: dict)`

`sqlalchemyseed.validator.check_model_key(entity: dict, entity_is_parent: bool)`

`sqlalchemyseed.validator.check_source_data(source_data, source_key: Key)`

`sqlalchemyseed.validator.check_source_key(entity: dict, source_keys: list) → Key`

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

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

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

S

- `sqlalchemyseed`, [15](#)
- `sqlalchemyseed.attribute`, [15](#)
- `sqlalchemyseed.constants`, [16](#)
- `sqlalchemyseed.dynamic_seeder`, [16](#)
- `sqlalchemyseed.errors`, [16](#)
- `sqlalchemyseed.json`, [17](#)
- `sqlalchemyseed.loader`, [18](#)
- `sqlalchemyseed.seeder`, [18](#)
- `sqlalchemyseed.util`, [19](#)
- `sqlalchemyseed.validator`, [20](#)

A

`AbstractSeeder` (class in `sqlalchemyseed.seeder`), 18
`attr_is_column()` (in module `sqlalchemyseed.attribute`), 15
`attr_is_relationship()` (in module `sqlalchemyseed.attribute`), 15
`attr_name` (`sqlalchemyseed.seeder.InstanceAttributeTuple` attribute), 19

B

`backward()` (`sqlalchemyseed.json.JsonWalker` method), 17

C

`check_attributes()` (`sqlalchemyseed.validator.SchemaValidator` method), 20
`check_data_type()` (in module `sqlalchemyseed.validator`), 20
`check_max_length()` (in module `sqlalchemyseed.validator`), 20
`check_model_key()` (in module `sqlalchemyseed.validator`), 20
`check_source_data()` (in module `sqlalchemyseed.validator`), 20
`check_source_key()` (in module `sqlalchemyseed.validator`), 20
`ClassNotFoundError`, 16
`current_key` (`sqlalchemyseed.json.JsonWalker` property), 17

D

`data()` (`sqlalchemyseed.validator.Key` class method), 20
`DATA_KEY` (in module `sqlalchemyseed.constants`), 16
`DynamicSeeder` (class in `sqlalchemyseed.dynamic_seeder`), 16
`DynamicSeeder` (class in `sqlalchemyseed.seeder`), 18

E

`EmptyDataError`, 16

`exec_func_iter()` (`sqlalchemyseed.json.JsonWalker` method), 17

F

`filter()` (`sqlalchemyseed.validator.Key` class method), 20
`FILTER_KEY` (in module `sqlalchemyseed.constants`), 16
`filter_kwargs()` (in module `sqlalchemyseed.seeder`), 19
`find_from_current()` (`sqlalchemyseed.json.JsonWalker` method), 17
`find_from_root()` (`sqlalchemyseed.json.JsonWalker` method), 17
`find_item()` (in module `sqlalchemyseed.util`), 19
`foreign_key_column()` (in module `sqlalchemyseed.attribute`), 15
`forward()` (`sqlalchemyseed.json.JsonWalker` method), 17

G

`generate_repr()` (in module `sqlalchemyseed.util`), 19
`get_model_class()` (in module `sqlalchemyseed.util`), 19
`get_model_class()` (`sqlalchemyseed.seeder.HybridSeeder` method), 19

H

`hybrid_validate()` (in module `sqlalchemyseed.validator`), 20
`HybridSeeder` (class in `sqlalchemyseed.seeder`), 19

I

`instance` (`sqlalchemyseed.seeder.InstanceAttributeTuple` attribute), 19
`InstanceAttributeTuple` (class in `sqlalchemyseed.seeder`), 19
`instances` (`sqlalchemyseed.seeder.AbstractSeeder` property), 18
`instances` (`sqlalchemyseed.seeder.HybridSeeder` property), 19
`instances` (`sqlalchemyseed.seeder.Seeder` property), 19

`instrumented_attribute()` (in module `sqlalchemyseed.attribute`), 15
`InvalidKeyError`, 16
`InvalidModelPath`, 16
`InvalidTypeError`, 16
`is_model()` (in module `sqlalchemyseed.util`), 19
`is_supported_class()` (in module `sqlalchemyseed.util`), 19
`is_valid_type()` (`sqlalchemyseed.validator.Key` method), 20
`iter_as_dict_items()` (`sqlalchemyseed.json.JsonWalker` method), 17
`iter_as_list()` (`sqlalchemyseed.json.JsonWalker` method), 17
`iter_kwargs_with_prefix()` (in module `sqlalchemyseed.util`), 20
`iter_non_ref_kwargs()` (in module `sqlalchemyseed.util`), 20
`iter_ref_kwargs()` (in module `sqlalchemyseed.util`), 20
`iterate_json()` (in module `sqlalchemyseed.util`), 20
`iterate_json_no_prefix()` (in module `sqlalchemyseed.util`), 20

J

`json` (`sqlalchemyseed.json.JsonWalker` property), 17
`json_is_dict` (`sqlalchemyseed.json.JsonWalker` property), 17
`json_is_list` (`sqlalchemyseed.json.JsonWalker` property), 17
`JsonWalker` (class in `sqlalchemyseed.json`), 17

K

`Key` (class in `sqlalchemyseed.validator`), 20
`keys()` (`sqlalchemyseed.json.JsonWalker` method), 18

L

`load_entities_from_csv()` (in module `sqlalchemyseed.loader`), 18
`load_entities_from_json()` (in module `sqlalchemyseed.loader`), 18
`load_entities_from_yaml()` (in module `sqlalchemyseed.loader`), 18

M

`MaxLengthExceededError`, 16
`MissingKeyError`, 16
`model()` (`sqlalchemyseed.validator.Key` class method), 20
`MODEL_KEY` (in module `sqlalchemyseed.constants`), 16
module
 `sqlalchemyseed`, 15
 `sqlalchemyseed.attribute`, 15

`sqlalchemyseed.constants`, 16
 `sqlalchemyseed.dynamic_seeder`, 16
 `sqlalchemyseed.errors`, 16
 `sqlalchemyseed.json`, 17
 `sqlalchemyseed.loader`, 18
 `sqlalchemyseed.seeder`, 18
 `sqlalchemyseed.util`, 19
 `sqlalchemyseed.validator`, 20

N

`NotInModuleError`, 17

P

`ParseError`, 17

R

`referenced_class()` (in module `sqlalchemyseed.attribute`), 15
`reset()` (`sqlalchemyseed.json.JsonWalker` method), 18

S

`SchemaValidator` (class in `sqlalchemyseed.validator`), 20
`seed()` (`sqlalchemyseed.seeder.AbstractSeeder` method), 18
`seed()` (`sqlalchemyseed.seeder.HybridSeeder` method), 19
`seed()` (`sqlalchemyseed.seeder.Seeder` method), 19
`Seeder` (class in `sqlalchemyseed.seeder`), 19
`set_instance_attribute()` (in module `sqlalchemyseed.attribute`), 15
`sort_json()` (in module `sqlalchemyseed.json`), 18
`SOURCE_KEYS` (in module `sqlalchemyseed.constants`), 16
`sqlalchemyseed`
 module, 15
`sqlalchemyseed.attribute`
 module, 15
`sqlalchemyseed.constants`
 module, 16
`sqlalchemyseed.dynamic_seeder`
 module, 16
`sqlalchemyseed.errors`
 module, 16
`sqlalchemyseed.json`
 module, 17
`sqlalchemyseed.loader`
 module, 18
`sqlalchemyseed.seeder`
 module, 18
`sqlalchemyseed.util`
 module, 19
`sqlalchemyseed.validator`
 module, 20

U

`UnsupportedClassError`, [17](#)

V

`validate()` (in module *sqlalchemyseed.validator*), [20](#)

`validate()` (*sqlalchemyseed.validator.SchemaValidator*
method), [20](#)