pytest-psqlgraph Documentation

Release 0.2.0rc2.dev1

Rowland Ogwara

Oct 22, 2021

CONTENTS

pytest-psqlgraph is a pytest plugin that provides a set of useful tools for testing applications that utilize psqlgraph.

CHAPTER

ONE

QUICK START

Install via `pip`

```
pip install pytest-psqlgraph
```

Define a `psqlgraph_config` fixture in `conftest.py`

```
from typing import Dict

import pytest
from psqlgraph.base import ORMBase, VoidedBase

from pytest_psqlgraph.models import DatabaseDriverConfig

active_dictionary = None # load dictionary to use for this connection
active_model = None # module containing all models to use for this connection

@pytest.fixture(scope="session")
def psqlgraph_config() -> Dict[str, DatabaseDriverConfig]:
   return dict(
      pg_driver=DatabaseDriverConfig(
         host="host",
         user="test",
         password="test",
         database="test",
         dictionary=active_dictionary,
         model=active_model,
         orm_base=ORMBase,
         extra_bases=[VoidedBase]
      )
   )
```

This will autogenerate a `pg_driver` fixture on demand.

```
def test_driver_initialized(pg_driver: psqlgraph.PsqlGraphDriver) -> None:
    """Tests fixture gets initialized correctly

    Tables are created and persistence happens
    Args:
        pg_driver (psqlgraph.PsqlGraphDriver): pg driver
    """
    assert pg_driver
```

(continues on next page)

(continued from previous page)

```
    with pg_driver.session_scope() as s:
        pg_driver.nodes().count()
```

run test

```
python -m pytest
```

CHAPTER

TWO

USER GUIDE

2.1 Getting Started

This describes how to quickly get started using pytest-psqlgraph

2.1.1 Step 1. Install

`pytest-flask` is available on PyPi, and can be easily installed via `pip`:

```
pip install pytest-psqlgraph
```

2.1.2 Step 2. Configure

Define your psqlgraph config fixture in `conftest.py`:

```
from typing import Dict

import pytest
from psqlgraph.base import ORMBase, VoidedBase

from pytest_psqlgraph.models import DatabaseDriverConfig

active_dictionary = None # load dictionary to use for this connection
active_model = None # module containing all models to use for this connection

@pytest.fixture(scope="session")
def psqlgraph_config() -> Dict[str, DatabaseDriverConfig]:
   return dict(
      pg_driver=DatabaseDriverConfig(
         host="host",
         user="test",
         password="test",
         database="test",
         dictionary=active_dictionary,
         model=active_model,
         orm_base=ORMBase,
         extra_bases=[VoidedBase]
      )
   )
```

Note: Automatic Fixture Generation.

While this fixture can be associated with any `scope`, it is recommended to use `session` scope. The key of the configuration will be used to auto generate a fixture that returns a psqlgraph connection instance. The fixture will created and delete tables during initialization and tear down respectively. One a per test basis, the fixture will truncate tables as part of the tear down process for a particular test.

2.1.3 Step 3. Write and Run Test

Now you can depend on the fixture `pg_driver` in your tests

2.1.4 What's next?

The *Features* section gives a more detailed view of available features, as well as test fixtures and markers.

Consult the pytest documentation for more information about pytest itself.

2.2 Features

2.2.1 Markers

`pytest-psqlgraph` registers the following markers. See the pytest documentation on pytest markers and for notes on using markers.

`pytest.mark.psqlgraph_data` - load test data

pytest.mark.**psqlgraph_data**(*name: str*, *driver_name: str*, *data_dir: str*, *resource: str*, *unique_key: str*, *mock_all_props: bool*, *post_processors*)

The mark used to pass options to your application config.

Parameters

- **name** (*str*) – The name of the variable injected into the test function that will hold the result of fixture.
- **driver_name** (*str*) – matching psqlgraph driver name
- **data_dir** (*str*) – A directory that holds all the data files used in test data generation
- **resource** (*str*) – name of the resource to load relative to the data_dir
- **unique_key** (*str*) – name of the property used for linking multiple nodes together. Defaults to `node_id`
- **unique_key** – Optional flag that specify how unspecified properties are generated. If True all node properties will be autogenerated
- **post_processor** – a collection of functions that will be executed once the nodes are generated

Return type list[psqgraph.Node]

Example usage:

```
# define a sample post action
def append_mr(node: psqlgraph.Node) -> None:
    """Appends Mr. to father's name"""
    node.name = "Mr. {}".format(node.name)

@pytest.mark.psqlgraph_data(
    name="pg_data",
    driver_name="pg_driver",
    data_dir=here,
    resource="sample.yaml",
    unique_key="node_id",
    mock_all_props=True,
    post_processors=[append_mr],
)
def test_pgdata_with_yaml(
    pg_driver: psqlgraph.PsqlGraphDriver, pg_data: List[psqlgraph.Node]
):
    """Tests use of pgdata to load initial from yaml/json"""

    assert len(pg_data) == 3
    with pg_driver.session_scope():
        node = pg_driver.nodes().get("father-1")
        assert node.name == "Mr. Samson 0."
```

2.3 Reference

Provides full reference to `pytest-psqlgraph`

2.3.1 Core Plugin

pytest_psqlgraph.plugin.**inject_driver_fixture**(*fixture: pytest_psqlgraph.helpers.DatabaseFixture*, *request: _pytest.fixtures.SubRequest*) → None

Resolves and setups psqlgraph driver fixtures based on psqlgraph_config entries

pytest_psqlgraph.plugin.**inject_marker_data**(*mark:* pytest_psqlgraph.models.PsqlgraphDataMark, *item: _pytest.python.Function*) → None

Resolves data for the custom psqlgraph data

Examples

```
@pytest.mark.psqlgraph_data(
  name="pg_data",
  driver_name="pg_driver",
  data_dir=here,
  resource="sample.yaml",
  post_processors=[append_mr],
)
```

(continues on next page)

(continued from previous page)

```
def test_example(pg_driver: psqlgraph.PsqlGraphDriver, pg_data: List[psqlgraph.
↪Node]) -> None:
  ...
```

pytest_psqlgraph.plugin.**pytest_collection_finish**(*session: _pytest.main.Session*) → None

A psqlgraph driver instance

Initializes the database tables and makes fixtures available .. rubric:: Example

code-block:

```
{"g": {
    "host": "localhost",
    "user": "test",
    "password": "test",
    "database": "test_db",
    "extra_bases": [],
    "models": model_module,
    "dictionary": dictionary instance
    }
}
```

2.3.2 models

class pytest_psqlgraph.models.**DataModel**(**args*, ***kwargs*)

Bases: Protocol

class pytest_psqlgraph.models.**DatabaseDriverConfig**(*host: str*, *user: str*, *password: str*, *database: str*, *model:* pytest_psqlgraph.models.DataModel, *dictionary:* pytest_psqlgraph.models.Dictionary, *package_namespace: Optional[str] = None*, *orm_base: Optional[sqlalchemy.ext.declarative.api.DeclarativeMeta] = None*, *extra_bases: Optional[Iterable[sqlalchemy.ext.declarative.api.DeclarativeMeta]] = None*, *globals: Optional[Dict[str, Any]] = None*)

Bases: object

psqlgraph database configuration data

host

postgres database hostname with port (if non default)

Type str

user

postgres database username

Type str

password

postgres database user password

Type str

database
postgres database name to connect to

Type str

package_namespace
optional parameter used to demarcate driver model classes

Type Optional[str]

model
The python module containing all the models associated with this database

Type *pytest_psqlgraph.models.DataModel*

dictionary
The instance containing the dictionary definitions

Type *pytest_psqlgraph.models.Dictionary*

orm_base
Optional sqlalchemy declarative base used by all models, this defaults to psqlgraph ORMBase

Type Optional[sqlalchemy.ext.declarative.api.DeclarativeMeta]

extra_bases
Iterable of bases that needs to be created/destroyed as part of the driver

Type Optional[Iterable[sqlalchemy.ext.declarative.api.DeclarativeMeta]]

globals
optional default property keys and values used for all nodes created

Type Optional[Dict[str, Any]]

class pytest_psqlgraph.models.**Dictionary**(**args*, ***kwargs*)
Bases: Protocol

A dictionary template

schema
node name and schema pairs

Type Dict[str, psqlgml.types.DictionarySchemaDict]

class pytest_psqlgraph.models.**MarkExtension**(*g: psqlgraph.psql.PsqlGraphDriver*)
Bases: object

An extension for psqlgraph data marker

post(*nodes: Iterable[psqlgraph.node.Node]*) → None
Same as pre, but executes after data has been persisted

Parameters **nodes** – all nodes generated and persisted

pre(*nodes: Iterable[psqlgraph.node.Node]*) → None
Executes just before the generated nodes are written to the database

Parameters **nodes** – list of nodes pull from the test data that will be written to the database

run(*node: psqlgraph.node.Node*) → None
Executes within the same transaction as the one that will write the current node

Parameters **node** – the current node just before it is added to the database

class pytest_psqlgraph.models.**PsqlgraphDataMark**(**args*, ***kwargs*)
Bases: dict

2.3.3 helpers

Helper functions

pytest_psqlgraph.helpers.**drop_tables**(*driver: pytest_psqlgraph.models.DatabaseDriver*) → None
Drops all tables in the listed orm_bases

pytest_psqlgraph.helpers.**truncate_tables**(*pg_driver: psqlgraph.psql.PsqlGraphDriver*) → None
Truncates all entries in the database

Parameters **pg_driver** – active driver

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX