pytest-dbt-core

Cor Zuurmond

Aug 25, 2023

CONTENTS:

Write unit tests for your dbt logic with *pytest-dbt-core*! *pytest-dbt-core* is a pytest plugin for testing your dbt projects.

CHAPTER

ONE

INSTALLATION

Install *pytest-dbt-core* via pip with

```
python -m pip install pytest-dbt-core
```

CHAPTER

TWO

USAGE

Create a macro:

```
{% macro normalize_column_names(column_names) %}
{%- set re = modules.re -%}

{%- for column_name in column_names -%}

    {%- set normalized_column_name = re.sub('[!?@#$()%&]', '', column_name).strip().
↪replace(' ', '_').replace('.', '_').rstrip('_').lower() -%}

    {# Columns should not start with digits #}
    {%- if normalized_column_name[0].isdigit()  -%}
        {% set normalized_column_name = '_' + normalized_column_name-%}
    {% endif -%}

    `{{ column_name }}` as {{ normalized_column_name }}
    {%- if not loop.last -%}, {% endif -%}

{%- endfor -%}

{% endmacro %}
```

Unit test a macro:

```
import pytest
from dbt.clients.jinja import MacroGenerator
from pyspark.sql import SparkSession


@pytest.mark.parametrize(
    "macro_generator",
    ["macro.dbt_project.normalize_column_names"],
    indirect=True,
)
@pytest.mark.parametrize(
    "column_name,expected_column_name",
    [
        ("unit", "unit"),
        ("column with spaces", "column_with_spaces"),
        ("c!h?a#r$a(c)t%e&rs", "characters"),
        ("trailing white spaces  ", "trailing_white_spaces"),
```

(continues on next page)

(continued from previous page)

```
        ("column.with.periods", "column_with_periods"),
        ("9leading number", "_9leading_number"),
        ("UPPERCASE", "uppercase"),
    ],
)
def test_normalize_column_names(
    spark_session: SparkSession,
    macro_generator: MacroGenerator,
    column_name: str,
    expected_column_name: str,
) -> None:
    """Test normalize column names with different scenarios."""
    normalized_column_names = macro_generator([column_name])
    out = spark_session.sql(
        f"SELECT {normalized_column_names} FROM (SELECT True AS `{column_name}`)"
    )
    assert out.columns[0] == expected_column_name, normalized_column_names
```

2.1 Configuration

The plugin runs in the context of a dbt project.

2.1.1 Project directory

When you run *pytest* from the root of your project, you do **not** need to set the project directory. If you want to run *pytest* from another location, you point the *–dbt-project-dir* option to the root of your project.

2.1.2 Target

If you want to use another target than the default, you set the *–dbt-target* option.

2.1.3 Profiles directory

If you want to change dbt's profiles directory, use the *–profiles-dir* option.

2.2 dbt-spark

dbt-spark users are recommend to use the Spark connection method when testing. Together with the pytest Spark plugin, a on-the-fly Spark session removes the need for hosting Spark.

2.2.1 Installation

Install *dbt-spark*, *pytest-dbt-core* and *pytest-spark* via pip with

```
python -m pip install dbt-spark pytest-dbt-core pytest-spark
```

2.2.2 Configuration

Configure *pytest-spark* via pytest configuration.

```
# setup.cfg
[tool:pytest]
spark_options =
    spark.executor.instances: 1
    spark.sql.catalogImplementation: in-memory
```

2.2.3 Usage

Use the *spark_session* fixture to set-up the unit test for your macro:

```
import pytest
from dbt.clients.jinja import MacroGenerator
from pyspark.sql import SparkSession


@pytest.mark.parametrize(
    "macro_generator", ["macro.spark_utils.get_tables"], indirect=True
)
def test_get_tables(
    spark_session: SparkSession, macro_generator: MacroGenerator
) -> None:
    """The get tables macro should return the created table."""
    expected_table = "default.example"
    spark_session.sql(f"CREATE TABLE {expected_table} (id int) USING parquet")
    tables = macro_generator()
    assert tables == [expected_table]
```

2.2.4 Test

Run the Pytest via your preferred interface.

```
pytest
```

2.3 Projects

The following projects use the *pytest-dbt-core* plugin:

- spark-utils

2.4 Indices and tables

- genindex
- modindex
- search