cassy - Python Cassandra Database API

Mathias Ammon

Aug 20, 2022

GETTING STARTED

Python Cassandra Database API.

Uses the excellent Hypermodern-Python project foundation proposed by Claudio Jolowicz.

CHAPTER

ONE

INSTALLATION

Following Sections provide overview on how to install the package.

Contents

1.1 Standard/User Install

Use the following advice to install the standard / user version of this package, once you have **at least one push** on your **main** and **develop** branch (so the respective *release workflows* are triggered).

1.1.1 Linux

Install using a console with your virtual environment activated:

Latest Stable Version

```
$ pip install cassy
```

Latest Development Version (potentially unstable)

```
$ pip install --index-url https://test.pypi.org/simple/ --extra-index-url https://pypi.
↪org/simple/ cassy
```

This installs the TestPyPI version of `cassy` while resolving the dependencies on PyPI.

1.2 Development Install

1. Install Pyenv (only if not already present)
2. Install Poetry and Nox (only if not already present)
3. Clone the repo to a local directory (uses package name if square bracket part is omitted):

```
$ git clone https://github.com/tZ3ma/cassy [cassy-develop]
```

4. Change to the new local repo folder and activate the desired python versions using pyenv:

```
$ ccd strutils

$ pyenv install 3.10.4 (adjust version to your needs)
$ pyenv install 3.9.13 (optional)
$ pyenv install 3.8.13 (optional)

$ pyenv local 3.10.4 3.9.13 3.8.13 (activate those desired)
```

5. Install the package with development requirements:

```
$ poetry install
```

56 Auto generate and activate a virtual environment where the installed package is installed:

```
$ poetry shell
```

7. (Optional) Alternatively, you can now run an interactive Python session, or the command-line interface if your package supports it:

```
$ poetry run python
$ poetry run cassy
```

CHAPTER

TWO

QUICK START

Following sections provide a - little talk, much code - introduction to cassy. Everything should be copy-pastable and work out of the box, given your *Installation* was successful.

CHAPTER

THREE

EXAMPLES

Following sections provide more sophisticated details on how to use this package. Everything should be copy-pastable and work out of the box, given your *Installation* was successful.

CHAPTER

FOUR

FAQ

Following sections try to provide a first clue on how to deal with commonly encountered issues.

CHAPTER

FIVE

WORKFLOWS (NOX AND POETRY)

Following sections provide information on how to use the excellent Hypermodern-Python project foundation proposed by Claudio Jolowicz

Contents

5.1 Documentation

This project uses Sphinx relying on docstrings in NumPy Style which get inforced by flake8-docstrings and darglint. Use Nox to conveniently build the documentaiton inside the `docs/_build` folder:

To tweak or add nox sessions, alter the `noxfile.py` inside this project's root directory.

5.1.1 nox -s docs – Statically build whats new

Build the documentation while only acutally rebuilding those files that changed:

```
nox -s docs
```

5.1.2 nox -s docs_rebuild – Statically build everything from scratch

Rebuild the entire documentation from scratch:

```
nox -s docs_rebuild
```

5.1.3 nox -s docs_live – Dynamically build from scratch (once)

Builts the documentation from scratch, servs it locally on port 8000, opens your default browser on the main page (`docs/_build/index.html`) and rebuilts any pages live, that are subject to change (when saved to disk).

Invaluable when creating the documentation!

```
nox -s docs_live
```

5.2 Testing

This project uses Nox to conveniently run both:

- unittests via the defacto standard pytest
- doctests via xdoctest

To tweak or add nox testing sessions, alter the `noxfile.py` inside this project's root directory.

5.2.1 nox -s tests – Unittests

Unittests reside in `tests/` inside the root directory of this project. Make sure to provide docstrings (since they are enforced, heh!) and add new test modules to `docs/source/unittests.rst`.

Run all unittests using nox:

```
nox -s tests
```

5.2.2 nox -r tests -- -m MARKER – Run specificly marked tests, excluded by default

Unittests can be marked by adding a `@pytest.mark.MARKER` decorator as for example in `tests/test_connectivity.py`:

```
@pytest.mark.con
def test_wikipedia_connectivity(request_random_wiki_article):
  """Try reaching the wikipedia site to get a random article."""
  answer = request_random_wiki_article
  print(answer)
  assert "Error" not in answer
```

These markers can be explicitly run by passsing the `-m MARKER` option to the nox session as in:

```
nox -s tests -- -m MARKER
```

This templates supports following markers by default:

- `con` – Marks interernet connection attempts
- `e2e` – Marks end 2 end tests
- `slow` - Marks particularly slow tests

These markers are **excluded** from the default `nox -s test` session (which also gets invoked by just calling `nox`). These are thus also excluded from the *Tests* CI-Workflow in `.github/workflows/tests.yml`. To modify this behavior or exclude additional markers modify the `"not e2e and not con and not slow"`, line inside the `noxfile.py`:

```
@nox.session(python="3.10")
def tests(session):
    """Run test suite."""
    args = session.posargs or [
        "--cov",
        "-m",
        "not e2e and not con and not slow",
        # append exlcuded markers as "and not ..."
    ]
    session.run("poetry", "install", "--no-dev", external=True)
    install_with_constraints(
        session,
        "coverage[toml]",
        "pytest",
        "pytest-cov",
        "pytest-mock",
    )
    session.run("pytest", *args)
```

So to test one of them run e.g.:

```
nox -s tests ---m con
```

5.2.3 nox -s xdoctests – Doctests

Me personally, I love doctests. I thinks they are the most natural form of testing. Since archiev both with them: enforced tests and pretty, copy-pastable examples inside your documentation.

Run all doctests using nox:

```
nox -s xdoctests
```

5.3 Committing

After new code is added and all tests are passed, following is the usua workflow:

1. Run Black to format your code `nox -s black`
2. Stage your changes using `git add`
3. Run the pre-commit session to test lint and format your package using `nox -s pre-commit`
4. Stage again to reflect changes done by pre-commit `git add`
5. Commit your changes using `git commit -m "MY MESSAGE"`

5.4 Realeasing and Publishing

This project template provides two major forms of automated publishing

1. Development 'release' publishes on TestPyPI
2. Stable release publishes on PyPI

5.4.1 Latest deveolpment publishes on TestPyPI

Pseudo release a (potentially unstable) development version of your package by Pushing or Merging a Pull-Request to your **remote develop branch**. This automatically triggers the *TestPyPI* Workflow in `.github/workflows/test-pypi`, which publishes a development version on TestPyPI.

To enable your repo interacting with your TestPyPI account you need to create an API-Token named `TEST_PYPI_TOKEN` in your TestPyPI account settings and declare it a Secret in your remote Github repo.

Assuming you've successfully generated and declared your Secret TestPyPI Api-Token, following workflow is proposed for creating a new (unstable) development release:

1. Add all changes to your **local develop branch**
2. Run the full test and lint suite using `nox`.
3. Commit and Push your changes to the **remote develop branch**.
4. The *TestPyPI* Workflow in `.github/workflows/test-pypi.yml` automatically publishes the package using Poetry using a dev versioning scheme.

5.4.2 Stable releases and publishes on PyPI

Release a stable version of your package by creating a Release of your **main/ master** branch via the Github website. This triggers the github Workflow called PyPI residing in `.github/workflows/pypi.yml`, which automatically creates a release on PyPI.

To enable your repo interacting with your PyPI account you need to create an API-Token named `PYPI_TOKEN` in your PyPI account settings and declare it a Secret in your remote Github repo.

Assuming you've successfully generated and declared your Secret PyPI Api-Token, following workflow is proposed for creating a new release:

1. Bump the package version on your **local develop branch** using `poetry version major|minor|patch|` following the Semantic-Versioning.
2. Run the full test and lint suite using `nox`.
3. Commit and Push your changes to the **remote develop branch**.
4. Create a Pull-Request from your **remote develop branch** to the **remote main / master** branch via your remote repo's github webpage.
5. Merge the Pull-Request on your remote repo using the github webpage
6. Create a Release using the remote repos webpage.

   Note that the *Release Drafter* Workflow in `.github/workflows/release-drafter.yml` automatically creates a release draft listing all your changes.
7. The *PyPI* Workflow in `.github/workflows/pypi.yml` automatically publishes the package using Poetry

5.5 Managing Dependencies

Project dependencies are managed using Poetry.

5.5.1 Adding

Adding third party dependencies is done by using the `poetry add` command.

poetry add PACKAGE – Adding required dependencies

Add a required third party package to your package by using poetry:

```
poetry add PACKAGE
```

poetry add --dev PACKAGE ^^ Adding additional developer dependencies

Add additional developer dependencies by using one of the following poetry commands:

```
poetry add --dev PACKAGE

poetry add package^1.0
poetry add "package>=1.0"
poetry add cassy@latest
```

(continues on next page)

(continued from previous page)

```
poetry add git+https://github.com/tZ3ma/cassy.git
poetry add git+https://github.com/tZ3ma/cassy.git#develop
poetry add ./my-package/
```

Adding local dependencies in editable mode

Modify the `pyproject.toml` file inside this project's root directory:

```
[tool.poetry.dependencies]
my-package = {path = "../my/path", develop = true}
```

Adding extras/optional dependencies

If the package(s) you want to install provide extras, you can specify them when adding the package by using one of the following lines:

```
poetry add requests[security,socks]
poetry add "requests[security,socks]~=2.22.0"
poetry add "git+https://github.com/pallets/flask.git@1.1.1[dotenv,dev]"
```

5.5.2 Updating

Updating third party dependencies is done by using the `poetry add` command.

nox update – Updating all dependencies

Update all project dependencies by using:

```
poetry update
```

nox update package1 package 2 – Updating explicit dependencies

Update specific dependencies by using:

```
poetry update package1 pakage2
```

5.5.3 Versioning

Bumping your package's verion is done by using the `poetry version semver` command. Where semver is one of poetry's supported Semantic-Versioning specifiers.

poetry version – Bump yout package version

To bump your package's version use one of the following poetry commands:

```
poetry add patch
poetry add minor
poetry add major
poetry add prepatch
poetry add preminor
poetry add premajor
poetry add prerelease
```

5.5.4 Removing

Removing third party dependencies is done by using the `poetry remove` command.

poetry remove – Remove third party dependencies

Remove a required third party package from your package by using poetry:

```
poetry remove PACKAGE
```

CHAPTER

SIX

CONTRIBUTOR GUIDE

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Contents

6.1 List of important resources for contributors

- Source Code
- Documentation
- Issue Tracker
- Code of Conduct

6.2 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

6.3 How to request a feature

Request features on the Issue Tracker.

6.4 How to set up your development environment

You need Python 3.7+ and the following tools:

- Poetry
- Nox

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run cassy
```

6.5 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the `tests` directory, and are written using the pytest testing framework.

6.6 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

CHAPTER

SEVEN

CODE OF CONDUCT

Contents

7.1 Our Pledge

We as members, contributors, and leaders pledge to try making participation in our community a harassment-free experience for everyone as we possibly can, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

7.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

7.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

7.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at mathias.ammon@tuhh.de. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

7.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

7.6.1 1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

7.6.2 2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

7.6.3 3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

7.6.4 4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

7.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

CHAPTER

EIGHT

AUTHORS

8.1 Lead Developers

Mathias Ammon

CHAPTER

NINE

API

cassy.

9.1 casdriv - Cassandra Driver Interface

Cassandra-Driver CRUD interface.

cassy.casdriv.**connect_session**(*ips=('127.0.0.1',)*, *port=9042*, ***kwargs*)

Connect and return session using `fogd_db.casdb`.

Parameters

- **ips** (*Container*) – Container holding contact_points to connect the cassandra.cluster.Cluster. Defaults to ("127.0.0.1",), which is the local host.
- **port** (*Number*) – The server-side port to open connections to. Defaults to 9042.
- **kwargs** – Additional arguments relegated to cassandra.cluster.Cluster

Returns Tuple of cassandra.session.Cluster and cassandra.session.Session as in `('cluster'=cluster, 'session'=session)`.

Return type *NamedTuple*

cassy.casdriv.**create_simple_keyspace**(*name*, *replication_factor*, *durable_writes=True*, *connections=None*)

Create a keyspace with SimpleStrategy for replica placement.

If the keyspace already exists, it will not be modified. Basically a very close wrapper of cassandra driver's cqlengine functionality

Parameters

- **name** (*str*) – name of keyspace to create
- **replication_factor** (*int*) – keyspace replication factor, used with SimpleStrategy
- **durable_writes** (*bool, default=True*) – Write log is bypassed if set to False.
- **connections** (*list*) – List of connection names.

cassy.casdriv.**create_entry**(*model*, *data*, *prior_syncing=False*, *keyspace=None*, *con=None*)

Create a new entry using a python data class model.

Parameters

- **model** – One of the cassandra python-driver data class models.
- **data** (*dict*) – Keyword value pairings of the data to be added. Most conform to the *model* used.

- **prior_syncing** (*bool, default=False*) – If `True` *synchronize_model()* is called before creation, to synchronize database table and model. (Required if you call `model.create` for the first time, or change any model attributes.
- **keyspace** (*str, None, default=None*) – String to specify the keyspace the table is created. If `None`, `model.__keyspace__` is used.
- **con** (*str, None, default=None*) – String to specify the connection name the table is created with. `casdriv` uses *cluster name* as default connection name.

  If `None`, `model.__connection__` is used.

Returns Instance of the *model* created using *data*.

Return type model class

`cassy.casdriv.`**delete_entry**(*model*, *primary_keys*, *values*, *keyspace=None*, *con=None*)

Delete an existing entry using a python data class model.

Parameters

- **model** – One of the cassandra python-driver data class models.
- **primary_keys** (*str, tuple*) – String or tuple of strings specifying the label/schema of the primary key(s) to be read.
- **values** (*str, tuple*) – String or tuple of strings specifying the value of the primary key to be queried for.
- **keyspace** (*str, None, default=None*) – String to specify the keyspace the table is created. If `None`, `model.__keyspace__` is used.
- **con** (*str, None, default=None*) – String to specify the connection name the table is created with. `casdriv` uses *cluster name* as default connection name.

  If `None`, `model.__connection__` is used.

`cassy.casdriv.`**get_all_entries**(*model*, *keyspace=None*, *con=None*)

Get all entries of an existing data class model table.

Uses the cassandra python-driver queries

Parameters

- **model** – One of the cassandra python-driver data class models.
- **keyspace** (*str, None, default=None*) – String to specify the keyspace the table is created. If `None`, `model.__keyspace__` is used.
- **con** (*str, None, default=None*) – String to specify the connection name the table is created with. `casdriv` uses *cluster name* as default connection name.

  If `None`, `model.__connection__` is used.

Returns

List of table entries created via a cassandra python-driver data class models.

Return type list

`cassy.casdriv.`**get_cluster_name**(*cluster*)

Return clusters registered name.

Parameters **cluster** (*cassandra.cluster.Cluster*) – cassandra.cluster.Cluster object holding the cassandra database.

Returns cluster.metadata.cluster_name

Return type str

cassy.casdriv.**list_keyspace_tables**(*cluster*, *keyspace*)

List all tables present in keyspaces.

Parameters

- **cluster** (*cassandra.cluster.Cluster*) – cassandra.cluster.Cluster object holding the cassandra database.
- **keyspace** (*str*) – String specifying the keyspace of the cluster of which all existing tables are to be listed

Returns List of strings specifying the tables currently present inside the *keyspace* of *cluster*.

Return type list

cassy.casdriv.**list_cluster_keyspaces**(*cluster*)

List all keyspaces present in cluster.

Parameters **cluster** (*cassandra.cluster.Cluster*) – cassandra.cluster.Cluster object holding the cassandra database.

Returns List of strings specifying the keyspaces currently present inside the *cluster*.

Return type list

cassy.casdriv.**list_table_primary_keys**(*cluster*, *keyspace*, *table*)

List all primary key labels (schemas?).

Parameters

- **cluster** (*cassandra.cluster.Cluster*) – cassandra.cluster.Cluster object holding the cassandra database.
- **keyspace** (*str*) – String specifying the keyspace of the *cluster* where *table* is to be found.
- **table** (*str*) – String specifying the table inside the *keyspace* of *cluster* of which the primary key lable(s) is(are) to be listed.

Returns List of strings specifying the found primary key lables

Return type list

cassy.casdriv.**list_table_columns**(*cluster*, *keyspace*, *table*)

List all primary key lables (schemas?).

Parameters

- **cluster** (*cassandra.cluster.Cluster*) – cassandra.cluster.Cluster object holding the cassandra database.
- **keyspace** (*str*) – String specifying the keyspace of the *cluster* where *table* is to be found.
- **table** (*str*) – String specifying the table inside the *keyspace* of *cluster* of which the column labels are to be listed.

Returns List of strings specifying the found column lables

Return type list

cassy.casdriv.**read_entry**(*model*, *primary_keys*, *values*, *keyspace=None*, *con=None*)

Read existing entry using a python data class model.

Uses the cassandra python-driver queries

Parameters

- **model** – One of the cassandra python-driver data class models.
- **primary_keys** (*str, tuple*) – String or tuple of strings specifying the label/schema of the primary key(s) to be read.
- **values** (*str, tuple*) – String or tuple of strings specifying the value of the primary key to be queried for.
- **keyspace** (*str, None, default=None*) – String to specify the keyspace the table is created. If None, `model.__keyspace__` is used.
- **con** (*str, None, default=None*) – String to specify the connection name the table is created with. `casdriv` uses *cluster name* as default connection name.

  If None, `model.__connection__` is used.

Returns Instance of the *model* read.

Return type model class

Raises **ValueError** – Raised when *primary_keys* is neither of type tuple or str.

cassy.casdriv.**synchronize_model**(*model*)

Synchronize the cassandra table with a python data class model.

Effectively creating a cassandra table out of a data class model, if not present.

Parameters **model** – One of the cassandra python-driver data class models. Python data class model to synch.

9.2 cql - Cassandra Query Language Interface

Cassandra Query Language interface.

cassy.cql.**create_keyspace**(*keyspace*, *session*, *replication='simple'*)

Create keyspace via session if neccessary.

cassy.cql.**get_cluster_name**(*session*)

Querry session for current cluster name.

Parameters **session** (*cassandra.cluster.Session*) – cassandra.session.Session Used to query the cluster for its name.

Returns Name of the cassandra cluster, the current session is connected to.

Return type str

cassy.cql.**list_column_values**(*cluster*, *session*, *keyspace*, *table*, *column*)

List all column values of table in keyspace.

Parameters

- **cluster** (*cassandra.cluster.Cluster*) – cassandra.cluster.Cluster object holding the cassandra database.

- **session** (*cassandra.cluster.Session, default=None*) – cassandra.session.Session used to query the table for clumn values.
- **keyspace** (*str*) – String specifying the keyspace of the *cluster* where *table* is to be found.
- **table** (*str*) – String specifying the table inside the *keyspace* of *cluster* of which the column labels are to be listed.
- **column** (*Number, str*) – Column specifier of which the values are to be listed.

Returns List of strings specifying the found column values

Return type list

Raises **KeyError** – Key Error raised if *keyspace*, *table* or *column* are unsuccesfully white-listed.

cassy.cql.**get_all_entries**(*cluster*, *session*, *keyspace*, *table*)

Get all rows of a table.

Parameters

- **cluster** (*cassandra.cluster.Cluster*) – cassandra.cluster.Cluster object holding the cassandra database.
- **session** (*cassandra.cluster.Session*) – cassandra.session.Session Use to execute query statement.
- **keyspace** (*str*) – String specifying the default keyspace the *table* is found.
- **table** (*str*) – String specifying the table queried.

Returns List of dictionairies of found entries.

Return type list

Raises **KeyError** – Key Error raised if *keyspace* or *table* are unsuccesfully white-listed.

cassy.cql.**drop_row**(*cns*, *keyspace*, *table*, *primary_key*, *value*)

Drop(delete) a cassandra table row.

Parameters

- **cns** (*NamedTuple*) – Tuple of cassandra.session.Cluster and cassandra.session.Session as in `('cluster'=cluster, 'session'=session)` and returned by *cassy.casdriv.connect_session()*.
- **keyspace** (*str*) – String specifying the default keyspace the *table* is found.
- **table** (*str*) – String specifying the table of which to drop a row.
- **primary_key** (*str*) – String specifying/identifying the primary key of the row to be dropped.
- **value** – *primary_key* value of the row to be dropped.

Raises **KeyError** – Key Error raised if *keyspace*, *table*, *primary_key*, *value* are unsuccesfully white-listed.

Examples

Dropping a row from the Martin Crawford database assuming a cluster and session were created using *cassy.casdriv.connect_session()*:

```
drop_row(
    cns=(cluster, session),
    keyspace="crawford",
    table="common_fruiting_trees",
    primary_key="latin",
    value="Prunus domestica",
)
```

cassy.cql.**drop_all_rows**(*cluster*, *session*, *keyspace*, *table*)

Drop(delete) an entire tables rows/content.

Parameters

- **cluster** (*cassandra.cluster.Cluster*) – cassandra.cluster.Cluster object holding the cassandra database.
- **session** (*cassandra.cluster.Session*) – cassandra.session.Session Use to execute query statement.
- **keyspace** (*str*) – String specifying the default keyspace the *table* is found.
- **table** (*str*) – String specifying the table of which to drop all rows.

Raises **KeyError** – Key Error raised if *keyspace* or *table* are unsuccesfully white-listed.

9.3 model - Dynamically Creatinig Hardcoded Data Models

Module fo dynamically creating hardcoded data models.

class cassy.model.**MetaModel**(*name: str*, *primary_keys: dict*, *clustering_keys: dict*, *columns: dict*)

Bases: object

Dynamically create a table model.

Parameters

- **name** (*str*) – String specifying the table name. Will be the name of the CQL table for this model.
- **primary_keys** (*dict*) – Dictionairy with primary keys as key and respective column data type as value as in

```
primary_keys = {"my_primary_key": "Text"}
```

- **clustering_keys** (*dict*) – Dictionairy with clustering keys as keys and `("Column Data Type", "clustering_order")` tuple specifying column data type and clustering order as in:

```
clustering_keys = {
    "my_clustering_key": ("Text", "ASC"),
    "my_other_clustering_key": ("Text", "DESC"),
}
```

- **columns** (*str, container*) – Dictionairy with column names as key and respective column data type as value as in

```
columns = {
    "my_attribute": "Text",
    "my_other_attribute": "Int",
    "my_third_attribute": "Blob",
}
```

See the DataStax documentation for available datatypes.

Examples

Default use case:

```
my_meta_model = MetaModel(
    name="CrawfordCommonFruitingTrees",
    primary_keys={"Latin": "Text"},
    clustering_keys={
        "English": ("Text", "ASC"),
        "German": ("Text", "ASC"),
    },
    columns={"USDA_Hardiness": "Integer"},
)
```

See also:

Cassandra Driver Data Model

cassy.model.**create_data_model**(*meta_model*, *path*, *overwrite=False*, *backup=True*)

Dynamically create a table model.

Creates a Datastax Cassandry Cqlenine Data Model

Parameters

- **meta_model** (*MetaModel*) – dataclass describing the model data.
- **path** (*pathlib.Path, str*) – Path or string specifying the folder the hardcoded module will be located.
- **overwrite** (*bool, default=False*) – Boolean indicating whether the hardcoded model data file should be overwritten. If *backup* is True, exsting file will be renamed to `existing-name_backup_hash(timestamp).py`
- **backup** (*bool, default=True*) – Boolean indicating whether the hardcoded model data file should be kept as backup in case of overwriting. Exsting file will be renamed to `existing-name_backup_hash(timestamp).py`

Returns Path the hardcoded data model file was created at.

Return type pathlib.Path

Examples

Default use case:

```
my_meta_model = MetaModel(
    name="CrawfordCommonFruitingTrees",
    primary_keys={"Latin": "Text"},
    clustering_keys={
        "English": ("Text", "ASC"),
        "German": ("Text", "ASC"),
    },
    columns={"USDA_Hardiness": "Integer"},
)

path = create_data_model(
    meta_model=my_meta_model,
    path=os.path.join("~", ".fogd.d", "my_model.py"),
    overwrite=True,
)
```

cassy.model.**retrieve_data_model**(*path*, *model_name*)

Retrieve previously hardcoded data model.

Parameters

- **path** (*pathlib.Path*, *str*) – Path or string specifying the folder the hardcoded module will be located.
- **model_name** (*str*) – String specifying the *MetaModel.name*.

Returns

Datastax Cassandry Cqlenine Data Model previously hardcoded in *path*.

Return type DataModel

Examples

Default use case:

```
model = retrieve_data_model(
    path=os.path.join("~", ".fogd.d", "pytest_home_model.py"),
    model_name="CrawfordCommonFruitingTrees",
)
```

CHAPTER

TEN

UNITTESTS

cassy's (pytest) package.

Cassy API Tests

10.1 Casdriv Testing

Test succesfull local cassandra db setup, used for further testing.

class tests.db_api.test_casdriv.**Plant**(***values*)

Bases: cassandra.cqlengine.models.Model

Plant Dataclass, ready to be cassandrad.

Parameters

- **latin** (*str*) – String specifying the latin plant name. Used as the primary key
- **english** (*str, Container*) – String, or container of strings, specifying the english trivial names.
- **german** (*str, Container*) – String, or container of strings, specifying the german trivial names.

exception DoesNotExist

Bases: cassandra.cqlengine.models.DoesNotExist

exception MultipleObjectsReturned

Bases: cassandra.cqlengine.models.MultipleObjectsReturned

class tests.db_api.test_casdriv.**ClusterPlant**(***values*)

Bases: cassandra.cqlengine.models.Model

Plant Dataclass, ready to be cassandrad.

Has additional clustering keys sorted in ascending order.

Parameters

- **latin** (*str*) – String specifying the latin plant name. Used as the primary key
- **english** (*str, Container*) – String, or container of strings, specifying the english trivial names.
- **german** (*str, Container*) – String, or container of strings, specifying the german trivial names.

exception DoesNotExist

Bases: cassandra.cqlengine.models.DoesNotExist

exception MultipleObjectsReturned

Bases: cassandra.cqlengine.models.MultipleObjectsReturned

tests.db_api.test_casdriv.**test_cluster_name**(*casdriv_cns*)

Test correct cluster name getting of casriv.

tests.db_api.test_casdriv.**test_keyspace_simple_creation**(*casdriv_cns*)

Test creating a keyspace using casdriv.

tests.db_api.test_casdriv.**test_list_cluster_keyspaces**(*casdriv_cns*)

Test casdriv list_cluster_keyspaces utility.

tests.db_api.test_casdriv.**test_create_entry**(*casdriv_cns*)

Test creating a db entry using create_entry.

tests.db_api.test_casdriv.**test_create_entry_without_syncing**(*casdriv_cns*)

Test creating a db entry using create_entry.

tests.db_api.test_casdriv.**test_create_cluster_entry**(*casdriv_cns*)

Test creating a db entry using create_entry and clustering keys.

tests.db_api.test_casdriv.**test_read_entry**(*casdriv_cns*)

Test creating a db entry using create_entry.

tests.db_api.test_casdriv.**test_read_entry_exception**(*casdriv_cns*)

Test creating a db entry using create_entry.

tests.db_api.test_casdriv.**test_list_primary_keys**(*casdriv_cns*)

Test creating a db entry using create_entry.

tests.db_api.test_casdriv.**test_list_table_columns**(*casdriv_cns*)

Test creating a db entry using create_entry.

tests.db_api.test_casdriv.**test_delete_entry**(*casdriv_cns*)

Test deleting a db entry using create_entry.

tests.db_api.test_casdriv.**test_get_all_entries**(*casdriv_cns*)

Test deleting a db entry using casdriv.delete_entry.

10.2 CQL Testing

Test succesfull local cassandra db setup, used for further testing.

tests.db_api.test_cql.**test_cluster_name**(*casdriv_cns*)

Test correct cluster name getting of casriv.

tests.db_api.test_cql.**test_keyspace_simple_creation**(*casdriv_cns*)

Test creating a keyspace using casdriv.create_keyspace.

tests.db_api.test_cql.**test_keyspace_replication_creation**(*casdriv_cns*)

Test casdriv.create_keyspace using replication argument.

tests.db_api.test_cql.**test_list_column_values**(*casdriv_cns*)
Test cql.list_column_values.

tests.db_api.test_cql.**test_list_column_values_exceptions**(*casdriv_cns*)
Test cql.list_column_values exception rasising.

tests.db_api.test_cql.**test_get_all_entries**(*casdriv_cns*)
Test cql.get_all_entries.

tests.db_api.test_cql.**test_get_all_entries_exceptions**(*casdriv_cns*)
Test cql.get_all_entries exception rasising.

tests.db_api.test_cql.**test_drop_row**(*casdriv_cns*)
Test cql.drop_row.

tests.db_api.test_cql.**test_drop_rows_exceptions**(*casdriv_cns*)
Test cql.list_column_values exception rasising.

tests.db_api.test_cql.**test_drop_all_rows**(*casdriv_cns*)
Test cql.drop_row.

tests.db_api.test_cql.**test_drop_all_rows_exceptions**(*casdriv_cns*)
Test cql.drop_all_rows exception rasising.

Default Template Tests

10.3 Connectivity Testing

Module for testing connectivity.

Meant to serve as template in case outgoing connections are to be tested.

tests.test_connectivity.**request_url**(*url*)
With wrapper to requests.get.

tests.test_connectivity.**request_random_wiki_article**()
Try reaching the wikipedia site to get a random article.

tests.test_connectivity.**test_wikipedia_connectivity**(*request_random_wiki_article*)
Try reaching the wikipedia site to get a random article.

10.4 End-To-End Testing

Module for end2end testing.

Meant to serve as template in case end2end tesing is sensible. Uses pytest markers to require manual test inclusion.

tests.test_end2end.**test_unix_specifics**()
Test unix specific things.

tests.test_end2end.**test_windows_specifics**()
Test windows specific things.

10.5 (Fake) API Testing

Module to test fake api usage.

Meant to serve as template in case the package uses non-local database.

class tests.test_fakeapi.**FakeAPI**

Bases: object

Fake API.

classmethod create()

Expensive operation to create API.

shutdown()

Expensive shutdown operation.

tests.test_fakeapi.**fake_api()**

Yield api inerface when needed.

Scope set to session, to only create once per test session.

Yields *FakeAPI* – FakeAPI instance using a localhost as url.

10.6 Http(s) Requests Testing

Module to test https request calls using the request package.

Targets are mostly mocked using pytest-mock plugin:

```
poetry add --dev pytest-mock
```

Meant to serve as template in case the package uses url based api calls.

tests.test_http_requests.**mock_requests_get**(*mocker*)

Fixture to mock reqeust.get calls.

tests.test_http_requests.**request_rnd_wiki_artcl()**

Try reaching the wikipedia site to get a random article.

tests.test_http_requests.**test_mock_gets_called**(*mock_requests_get*, *request_rnd_wiki_artcl*)

Assert the requests.get was actually called.

random_wiki_article gets called by fixutre wrap around mock object so the mock object is "requests.get" instead of the url. Fixture order is important!

Since the data content is not of importance calling the fixture alone suffices.

Parameters

- **mock_requests_get** – mock_requests_get fixture from above
- **request_rnd_wiki_artcl** – request_random_wiki_article fixture from above

tests.test_http_requests.**test_mock_result_inspection**(*mock_requests_get*, *request_rnd_wiki_artcl*)

Test successful mock result inspection.

tests.test_http_requests.**test_mock_param_call_inspection**(*mock_requests_get*, *request_rnd_wiki_artcl*)

Assert the requests.get was called properly.

Random_wiki_article gets called by fixutre wrap around mock object so the mock object is "requests.get" instead of the url. Fixture order is implortant! Since the data content is not of importance calling the fixture alone suffices.

Parameters

- **mock_requests_get** – mock_requests_get fixture from above
- **request_rnd_wiki_artcl** – request_random_wiki_article fixture from above

tests.test_http_requests.**test_fail_on_request_error**(*mock_requests_get*, *request_rnd_wiki_artcl*)

Test on failing the https request.

10.7 Version Testing

Examplary test package to test version related issues.

tests.test_version.**test_verssion_access**()

Test for correct package version.

CHAPTER

ELEVEN

CHANGELOG

See the Github Releases webpage for what is new :)

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search
- Glossary

PYTHON MODULE INDEX

c

t

INDEX