valarpy

Release 3.0.0

[, ', D, o, u, g, l, a, s, , M, y, e, r, s, -, T, u, r, n, b, u, l, l, ',]

Aug 11, 2022

CONTENTS

CHAPTER

ONE

SETUP

First, install valarpy with just `pip install valarpy`. Note that if you are using sauronlab, you only need to install sauronlab.

1.1 Tunnel into Valinor

Valarpy connects to Valar through an SSH tunnel; the database is not accessible remotely. There are two modes of connection: Valarpy can either use an existing SSH tunnel or create its own.

Replacing *53419* with a number of your choosing, The port can't be *anything*. It needs to be between 1025 and 65535, and I recommend 49152–65535.

create the tunnel using:

```
ssh -L 53419:localhost:3306 valinor.ucsf.edu
```

Note that after running it your shell is now on Valinor.

You will need to leave this tunnel open while connecting to Valar. As long the terminal window connection is open, you can access valar through your notebooks.

You can of course alias in your `~/.commonrc`. Adding these lines will provide a `valinor-tunnel` alias:

```
export valinor_tunnel_port=53419
alias valinor-tunnel='ssh -L ${valinor_tunnel_port}:localhost:3306 valinor.ucsf.edu'
```

1.2 Connect to Valar

You will connect to Valar via that tunnel. An example configuration file is provided in the readme. I recommend downloading it to `$HOME/.sauronlab/connection.json`. You'll need to fill in the username and password for the database connection. Although the database users are provided for safety (no write access by default) rather than security, do not put a username and/or password anywhere that's web-accessible (including GitHub).

CHAPTER

TWO

USAGE

Valarpy is an ORM for Valar based on Peewee. Here is how to initiate a new (read-only) connection.

```
import valarpy
with valarpy.opened() as model:
    print(list(model.Refs.select()))
```

2.1 Write access

If your database username provides privileges for `INSERT`, `UPDATE`, or `DELETE`, you can run those queries via valarpy. Because the vast majority of access does not require modifying the database – and mistakes can be catastrophic and require reloading from a nightly backup – you must call `enable_write()`.

Warning: Although valarpy is mostly thread-safe when using atomic transactions, it uses a global write-access flag. That means that if you call `enable_write` (see below), all code can write, even if it called `valarpy.opened()` separately.

You should use transactions and/or savepoints. In the following code, an atomic transaction is started, and the transaction is committed when the context manager closes. If an exception is raised within the transaction block, it will be automatically rolled back on exit.

```
import valarpy
with valarpy.opened() as model:
    model.conn.backend.enable_write()
    with model.atomic():
        ref = Refs.fetch(1)
        ref.name = "modified-name"
        ref.save()
    # transaction is now committed
```

If you nest `atomic()` calls, the nested call(s) will create savepoints rather than initiate new transactions. If a transaction fails in the nested block, it will be rolled back to the savepoint:

```
import valarpy
with valarpy.opened() as model:
    # enable write access
    model.conn.backend.enable_write()
    # This starts a transaction:
```

(continues on next page)

(continued from previous page)

```
    with model.conn.atomic():
        ref = Refs.fetch(1)
        ref.name = "improved-name"
        ref.version = "version 2"
        try:
            # This just creates a savepoint
            with model.conn.atomic():
                ref.name = "name modified again"
                ref.save()
                raise ValueError("Fail!")
            # savepoint exits
        except:
            print("Rolling back to checkpoint")
            # prints "name=improved-name, version="version 2"
            print(f"name={ref.name}, version={ref.version}")
    # transaction is now committed
```

You would never nest `atomic` calls in a single function, but you might call a function that also calls `atomic`. There is a method analogous to `atomic` called `rolling_back`. This will roll back to the transaction or savepoint when the block closes, whether or not an exception was called. This is especially useful when writing tests. Finally, `model.atomic()` and `model.rolling_back()` both yield a Transaction object that has several methods, including `.commit()` and `.rollback()`. In general, you would not want to call these directly, but you can.

CHAPTER

THREE

NOTES ABOUT TABLES

Assay frames and features (such as MI) are stored as MySQL binary `blob`s.

Each frame in `assay_frames` is represented as a single big-endian unsigned byte. To convert back, use `utils.blob_to_byte_array(blob)`, where `blob` is the Python `bytes` object returned directly from the database.

Each value in `well_features` (each value is a frame for features like MI) is represented as 4 consecutive bytes that constitute a single big-endian unsigned float (IEEE 754 `binary32`). Use `utils.blob_to_float_array(blob)` to convert back.

There shouldn't be a need to insert these data from Python, so there's no way to convert in the forwards direction.

CHAPTER

FOUR

API REFERENCE

This page contains auto-generated API reference documentation[1].

4.1 valarpy

Project metadata and convenience functions.

4.1.1 Package Contents

valarpy.**new_model**()

Shorthand for importing model. You should have a connection open.

Returns
The `model` module

valarpy.**opened**(*config: Union[None, str, pathlib.Path, List[Union[str, pathlib.Path, None]], Mapping[str, Union[str, int]]] = None*)

Context manager. Opens a connection and returns the model. Closes the connection when the generator exits.

Parameters
config – Passed to `Valar.__init__`

Yields
The `model` module, with an attached `.conn` of type `Valar`

valarpy.**valarpy_info**() → Generator[str, None, None]

Gets lines describing valarpy metadata and database row counts. Useful for verifying that the schema matches the valarpy model, and for printing info.

Yields
Lines of free text

Raises
InterfaceError – On some connection and schema mismatch errors

Python code to talk to Valar. There is more documentation available in that readme.

[1] Created with sphinx-autoapi

PYTHON MODULE INDEX

INDEX