03_Indexing_S3_MDFs_With_PonyORM-FancyClasses

April 21, 2020

1 Indexing MDFs with Pony ORM.

Continues from the SQLLite PonyORM example to make a more advanced MDF class to display more information in the `__repr__` string.

[1]:
```
import fsspec
from asammdf import MDF
```

ORM setup.

[2]:
```
import os

import pony.orm
from pony.orm.core import EntityMeta
from datetime import datetime

pony.orm.set_sql_debug(False)
db = pony.orm.Database()

if True:
    # In memory datatabase
    filename=":memory:"
else:
    # Or not.
    filename = os.path.abspath("mdf_index.sqlite")
    if os.path.exists(filename):
        os.unlink(filename)
# Bind
db.bind(
    provider="sqlite", filename=filename, create_db=True,
)
```

Rather than calculate the product and company name every time, this stores them in the database.

The repr string is also more descriptive.

[3]:
```
# For Local Indexing.
class MDF(db.Entity):
```

```
    # Filesystem Bits.
    key = pony.orm.Required(str, unique=True,)
    last_modified = pony.orm.Optional(datetime, volatile=True)
    etag = pony.orm.Optional(str,)
    size = pony.orm.Optional(int,)
    size_mb = pony.orm.Optional(float,)
    storage_class = pony.orm.Optional(str,)
    type = pony.orm.Optional(str,)
    name = pony.orm.Optional(str,)

    # Pre-calculated bits.
    basename = pony.orm.Optional(str,)

    product = pony.orm.Optional(str,)
    company = pony.orm.Optional(str,)

    # ASAM MDF Bits.
    version = pony.orm.Optional(str,)
    channels = pony.orm.Set("Channel",)

    def __repr__(self):
        return f"MDF<{self.id},{self.product},{self.company},Ch:{len(self.
↪channels)}>"

class Channel(db.Entity):
    """Channel entity to represent a

    """
    name = pony.orm.Required(str, unique=True,)
    mdfs = pony.orm.Set("MDF",)

    def __repr__(self):
        return f"Channel<{self.id},{self.name}>"

def upsert(cls, get, set=None):
    """
    Interacting with Pony entities.

    :param cls: The actual entity class
    :param get: Identify the object (e.g. row) with this dictionary
    :param set: Additional fields to set if ```get``` returns nothing.
    :return:
    """
    # does the object exist
    assert isinstance(cls, EntityMeta), f"{cls} is not a database entity"

    # if no set dictionary has been specified
```

```
    set = set or {}

    if not cls.exists(**get):
        # make new object
        return cls(**set, **get)
    else:
        # get the existing object
        obj = cls.get(**get)
        for key, value in set.items():
            obj.__setattr__(key, value)
        return obj


db.generate_mapping(create_tables=True)
```

Indexing functions.

[4]:
```
def index_mdf(mdf_path):
    """ Index the mdf file itself. """
    info = fs.info(mdf_path)
    # Local File
    MDF_ = upsert(
    cls=MDF,
    get={"key": info["Key"]},
    set={
        "last_modified": info["LastModified"],
        "etag": info["ETag"],
        "size": info["size"],
        "size_mb": info["size"] / 1024 ** 2,
        "storage_class": info["StorageClass"],
        "type": info["type"],
        "name": info["name"],
        "basename": os.path.basename(info["name"])
        },
    )
    try:
        db.commit()
        return MDF_
    except:
        db.rollback()
        return None
import asammdf
def index_channels(mdf):
    """Given a MDF files, process the channels

    """
    # Open the MDF file.
```

```
    with fs.open(mdf.name, "rb") as fid:
        mdf_ = asammdf.MDF(fid)
    #
    channels=list()
    # Loop through each of the channels in the database.
    for channel in mdf_.channels_db.keys():
        print(".", end="")
        channel_ = upsert(Channel, {"name": channel})
        channels.append(channel_)
    print("")
    MDF_ = upsert(
    cls=MDF,
    get={"name": mdf.name},
    set={
        "channels": channels
        },
    )
    try:
        db.commit()
        return channels
    except:
        db.rollback()
        return None

def index_mdf_info(mdf):
    """ Index company and product information in the database from the filename.
 ↪"""
    product = os.path.basename(os.path.dirname(mdf.name))
    company = os.path.basename(
        os.path.dirname(
            os.path.dirname(
                mdf.name
            )
        )
    )
    # Local File
    MDF_ = upsert(
    cls=MDF,
    get={"name": mdf.name},
    set={
        "product": product,
        "company": company,
        },
    )
    try:
        db.commit()
        return MDF_
```

```
    except:
        db.rollback()
        return None
```

[5]:
```
import os
import random
mdf_paths=list()



s3_cfg = {
    "key": "mdf_minio_access_key",
    "secret": "mdf_minio_secret_key",
    "client_kwargs": {
        "endpoint_url": "http://minio:9000",
    },
}

fs = fsspec.filesystem("s3", **s3_cfg)
for bucket in fs.ls(""):
    for root, dirs, files in fs.walk(bucket):
        for file in files:
            if file.lower().endswith(".mf4") or file.lower().endswith(".mdf"):
                mdf_paths.append(os.path.join(root, file))
print(f"Found {len(mdf_paths)} MDF files")
```

```
Found 975 MDF files
```

Randomly pick a file for analysis.

[6]:
```
mdf_path = random.choice(mdf_paths)
mdf_path
```

[6]:
```
'mdfbucket-5/DanishStartup/BoatyMcBoatface/107af0de-a47b-47a2-8af3-0670064f0519.
mf4'
```

Insert the MDF file into the database.

[Notice the **repr** string isn't fully populated, the data isn't yet in the database]

[7]:
```
mdf = index_mdf(mdf_path)
mdf
```

[7]:
```
MDF<1,,,Ch:0>
```

Index the product and company name of the mdf

[8]:
```
index_mdf_info(mdf)
mdf
```

[8]:
```
MDF<1,BoatyMcBoatface,DanishStartup,Ch:0>
```

Index the channels.

[9]:
```
index_channels(mdf)
mdf
```

…

[9]:
```
MDF<1,BoatyMcBoatface,DanishStartup,Ch:13>
```

[10]:
```
%%timeit
for mdf_path in mdf_paths[:10]:
    index_mdf(mdf_path)
```

```
4.67 ms ± 12.7 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

[11]:
```
pony.orm.select(m for m in MDF)[0:10]
```

[11]:
```
[MDF<1,BoatyMcBoatface,DanishStartup,Ch:13>, MDF<2,,,Ch:0>, MDF<3,,,Ch:0>,
MDF<4,,,Ch:0>, MDF<5,,,Ch:0>, MDF<6,,,Ch:0>, MDF<7,,,Ch:0>, MDF<8,,,Ch:0>,
MDF<9,,,Ch:0>, MDF<10,,,Ch:0>]
```

[12]:
```
%%timeit
for mdf in pony.orm.select(m for m in MDF)[0:10]:
    index_mdf_info(mdf)
```

```
9.36 ms ± 61.9 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

[13]:
```
pony.orm.select(m for m in MDF)[0:10]
```

[13]:
```
[MDF<1,BoatyMcBoatface,DanishStartup,Ch:13>,
MDF<2,BoatyMcBoatface,ABMøøse,Ch:0>, MDF<3,BoatyMcBoatface,ABMøøse,Ch:0>,
MDF<4,Bulldozer,ABMøøse,Ch:0>, MDF<5,Bulldozer,ABMøøse,Ch:0>,
MDF<6,Bulldozer,ABMøøse,Ch:0>, MDF<7,Car,ABMøøse,Ch:0>, MDF<8,Car,ABMøøse,Ch:0>,
MDF<9,DumpTruck,ABMøøse,Ch:0>, MDF<10,DumpTruck,ABMøøse,Ch:0>]
```

[14]:
```
%%timeit
for mdf in pony.orm.select(m for m in MDF)[0:10]:
    index_channels(mdf)
```

…
…
…
…
…
…
…
…

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
234 ms ± 30.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

This task can easily be distributed with celery or rq

Asynchronous Task Execution In Python

[15]:
```
for mdf_path in mdf_paths:
    mdf = index_mdf(mdf_path)
    index_mdf_info(mdf)
    index_channels(mdf)
```

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

...

2 Using Indexed Data

[16]:
```
channels = pony.orm.select(c for c in Channel)
```

[17]:
```
for channel in channels:
    break
```

How many MDF files have been indexed?

[18]:
```
len(channel.mdfs)
```

[18]: 975

How many bytes of MDF files have been indexed?

[19]:
```
pony.orm.sum(m.size for m in MDF)
```

[19]: 2301175672

How many GB of MDF files have been indexed?

[20]:
```
pony.orm.sum(m.size for m in MDF)/1024**3
```

[20]: 2.143136851489544

Find the biggest MDF file to analyze:

[21]:
```
q = pony.orm.select(mdf for mdf in MDF).order_by(lambda: pony.orm.desc(mdf.
↪size))
```

[22]:
```
q[0:5]
```

[22]:
```
[MDF<28,BoatyMcBoatface,DanishStartup,Ch:13>,
MDF<47,Bulldozer,DäsCarGmbh,Ch:13>,
MDF<136,BoatyMcBoatface,DanishStartup,Ch:13>, MDF<190,BoatyMcBoatface, ,Ch:13>,
MDF<212,Car,ABMøøse,Ch:13>]
```

[23]:
```
fid = fs.open(q.first().name)
mdf_ = asammdf.MDF(fid)
```

[24]:
```
%matplotlib inline
```

[25]:
```
import matplotlib.pyplot as plt
```

[26]:
```
import numpy as np
```

[27]:
```
t = np.where(chan.timestamps<1)
plt.plot(chan.timestamps[t], chan.samples[t])
```

```
␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-27-f3f19fad5b36> in <module>
----> 1 t = np.where(chan.timestamps<1)
      2 plt.plot(chan.timestamps[t], chan.samples[t])


NameError: name 'chan' is not defined
```