Как и зачем делать свой ORM на Python

Serge Matveenko
DataArt

github.com/lig

What is ORM?

- ORM stays for “Object-relational mapping”
- ODM (Object-document mapping) is ORM too
- Helps to persist objects no matter what is under the hood
- Helps to build complex queries
- Knows what the data scheme is looks like
- Could help to maintain the DB layer (Code to DB)
- Could reflect the DB schema (DB to Code)
- Could help to cache data

What we have in Python

- SQLAlchemy — very powerful, hard to learn syntax
- DjangoORM — powerful enough, easier to learn syntax
- PonyORM — not that powerful, awesome syntax
- Peewee — SQL powerful, SQL inspired syntax with cookies
- MongoEngine — Django like ORM for MongoDB, good for start

SQLAlchemy

```
class Person(Base):
    __tablename__ = 'person'
    id = Column(Integer, primary_key=True)
    name = Column(String(250), nullable=False)

engine = create_engine('sqlite:///sqlalchemy_example.db' )
Base.metadata.bind = engine
DBSession = sessionmaker(bind=engine)
session = DBSession()

new_person = Person(name='new person')
session.add(new_person)
session.commit()

person = session.query(Person).first()
```

DjangoORM

```python
class Person(Model):
    name = CharField(max_length=250)

new_person = Person(name='new person')
new_person.save()

person = Person.objects.first()
```

PonyORM

```python
class Person(db.Entity):
    name = Required(str)

with db_session:
    new_person = Person(name='new person')

person = select(p for p in Person)[0]
```

Custom ORM

- ORM is more than mapping
- Any Data Schema representation
- External Data Validation
- Data Processing
- Serialization/Deserialization
- Awesome way to use Python

Everytime you write ORM

Disclaimer:
just kidding :)

Guido becomes a bit happier

A typical ORM

```python
class Author(Model):
    name = CharField()

class Book(Model):
    title = CharField()
    year = IntField()
    author = Relation(Author, 'books')

william_gibson = Author(name='William Gibson')
count_zero = Book(title='Count Zero', year=1986, author=william_gibson)

gibsons_books = william_gibson.books
```

Basic Field (simple descriptor)

```
class Field:

    def __get__(self, obj, type=None):
        return obj._data[self._name]

    def __set__(self, obj, value):
        obj._data[self._name] = value
```

Basic Field (machinery)

```
class ModelMeta(type):
    def __new__(cls, name, bases, attrs):
        for field_name, field in attrs.items():
            field._name = field_name
        attrs['_data'] = StrictDict.fromkeys(attrs.keys())
        return type(name, bases, attrs)

class Model(metaclass=ModelMeta):
    pass

class Field:
    def __get__(self, obj, type=None):
        return obj._data[self._name]
    def __set__(self, obj, value):
        obj._data[self._name] = value
```

Simple Validation

```
class CharField(Field):

    def __set__(self, obj, value):

        if not isinstance(value, str):
            raise TypeError(obj, self._name, str, value)

        super().__set__(obj, value)
```

Relation

```
class Relation(Field):

    def __init__(self, rel_model_class):
        self._rel_model_class = rel_model_class

    def __set__(self, obj, value):

        if not isinstance(value, self._rel_model_class):
            raise TypeError(obj, self._name, self._rel_model_class, value)

        super().__set__(obj, value)


class Book(Model):
    author = Relation(Author)
```

Reverse Relation

```
class Author(Model):
    name = CharField()


class Book(Model):
    author = Relation(Author, 'books')


william_gibson = Author(name='William Gibson')

gibsons_books = william_gibson.books
```

Reverse Relation

```
class Relation(Field):
    def __init__(self, rel_model_class, reverse_name):
        self._rel_model_class, self._reverse_name = rel_model_class,
reverse_name

class ReverseRelation:
    def __init__(self, origin_model, field_name):
        self._origin_model, self._field_name = origin_model, field_name
    def __get__(self, obj, type=None):
        return self._origin_model.S.filter(self._field_name=obj)

class ModelMeta(type):
  def __new__(cls, name, bases, attrs):
    type_new = type(name, bases, attrs)
    for field_name, field in attrs.items():
        if isinstance(field, Relation):
        setattr(field._rel_model_class, self._reverse_name,
                ReverseRelation(type_new, field_name))
    return type_new
```

Just a Validation

```python
class ValidatorMeta(type):
    def __call__(cls, **attrs):
        for attr_name, attr in attrs.items():
            if not isinstance(attr, getattr(cls, attr_name)):
                raise TypeError()
        return dict(**attrs)

class Validator(metaclass=ValidatorMeta):
    pass

class FooBar(Validator):
    foo = str
    bar = int

FooBar(foo='spam', bar=42) == {'bar': 42, 'foo': 'spam'}
```

Learn Python magic

- Meta classes
- Descriptors
- Class attributes
- Python data model
- `object.__new__`
- `type.__call__`
- `type.__prepare__`
- `object.__instancecheck__`

Thank you!

github.com/lig