Implementing git-orm

https://github.com/natano/python-git-orm/

Martin Natano

June 06, 2013

whoami

Martin Natano

- Web Developer @RadarServices
- previously: Medical University of Vienna, Mjam

Description

- django-esque model interface for storing objects in a git repository
- ISC licensed
- written in python

Models

Models

Listing 1: Python

```python
from git_orm import models

class User(models.Model):
    email = models.TextField(primary_key=True)
    name = models.TextField(null=True)

class Article(models.Model):
    author = models.ForeignKey(User)
    summary = models.TextField()
    content = models.TextField()
```

Creating an object

Listing 2: Python

```
hansel = User()
hansel.email = 'hansel@example.com'
hansel.name = 'Hans'
hansel.save()

# or

gretel = User.create(email='grete@example.com',
                     name='Grete')
```

Timestamps

Listing 3: Python

```
>>> gretel.created_at
datetime.datetime(2013, 6, 6, 12, 27, 35, 276071)
>>> gretel.updated_at
datetime.datetime(2013, 6, 6, 10, 08, 33, 3)
```

Querysets

Expressive Syntax

Listing 4: Python

```
User.objects.get(email='user@example.org')
User.objects.exists()
Article.objects.all()
Article.objects.count()
Article.objects.filter(author_email='user@example.org')
Article.objects.exclude(summary__contains='vienna.rb')
```

Chaining

Listing 5: Python

```
(Article.objects
     .exclude(author_email='user@example.org')
     .filter(summary__icontains='vienna.rb')
     .count())
User.objects.order_by('email')[:10]
```

Lazy Evaluation

Listing 6: Python

```
articles = Article.objects.all()
articles = articles.filter(published_at__lt=now)
articles.count()
```

Q Objects

Q Objects (Advanced Querying)

Listing 7: Python

```python
from git_orm.models import Q

User.objects.filter(
    Q(email__endswith='@example.org')
    |Q(name__icontains='grete')
)
```

Q Objects (connect 'em)

Listing 8: Python

```python
from git_orm.models import Q

~Q(name='Hexe') & (
    Q(email='grete@example.com') | Q(name='Hansel'))
```

Transactions

Automatic Transaction Management

Listing 9: Python

```python
from git_orm import transaction

with transaction.wrap():
    ...
```

Automatic Transaction Management

Listing 10: Python

```
@transaction.wrap()
def persist():
    ...
```

Manual Transaction Management

Listing 11: Python

```python
from git_orm import transaction

transaction.begin()
...
transaction.commit()

transaction.begin()
...
transaction.rollback()
```

Syntactic Sugar

Context Managers

Context Managers

Listing 12: Python

```
with transaction.wrap() as trans:
    ...
```

Listing 13: Ruby

```
Transaction.wrap do |trans|
    ...
end
```

Context Managers

Listing 14: Python

```
class wrap:
    ...
    def __enter__(self):
        begin()
        return _transaction

    def __exit__(self, type, value, traceback):
        if not type and _transaction.has_changes:
            commit(self.message)
        else:
            rollback()
    ...
```

Decorators

Decorators

Listing 15: Python

```
@transaction.wrap()
def persist:
    ...
```

Listing 16: Ruby

```
def persist
    # ??
end
```

Decorators

Listing 17: Python

```python
class wrap:
    ...
    def __call__(self, fn):
        @wraps(fn)
        def _inner(*args, **kwargs):
            with self:
                return fn(*args, **kwargs)
        return _inner
    ...
```

Operator Overloading

Operator Overloading

Listing 18: Python

```
Q(...) & Q(...) | Q(...)
```

Listing 19: Ruby

```
Q(...) & Q(...) | Q(...)
```

Metaclasses

Metaclasses

Listing 20: Python

```
class Foo(object):
    __metaclass__ = Bar
```

Listing 21: Ruby

```
class Foo
    extend Bar
end
```

Bugs & Shortcomings

Bugs & Shortcomings

- python 3 only
- delete not implemented yet
- concurrent transactions are not handled correctly

Up next

Up next & Ideas

- python 2 & 3 support with six
- merging of concurrent transactions
- more query optimizations

Thx!

```
$ pip install git-orm
```