MoSQL

Mosky

More than SQL, but Less than ORM

MoSQL

Outline

Why not SQL?

SQL Syntax

- SELECT * FROM article;
- SELECT * FROM article LIMIT 1;
- add “ ORDER BY created ”?
- add “ OFFSET 10 ”?
- add “ GROUP BY author ”?
- Is “ UPDATE article WHERE title='SQL' SET title='ORM' ” correct?

!@#$%

SQL Injection

- ') or '1'='1
- ' or true; --
- ' or 1=1; --
- ' or 2=2; --
- ' or 'str'='str'; --
- …

It may be hacker friendly.

SQL seems ancient, but ...

using SQL is the FASTEST way.

Why ORM?

ORM Syntax

```
class User(Base):
    __tablename__ = 'users'
    name = Column(String)
    fullname = Column(String)
    password = Column(String)
```

ORM Syntax (cont.)

```
>>> fake_user = User('fakeuser', 'Invalid',
'12345')

>>> session.add(fake_user)


>>> for row in session.query(User,
User.name).all():

...     print row.User, row.name
```

hmmm …

SQL Injection

- ~~' or true; --~~
- ~~' or 1=1; --~~
- ~~' or 1=1; #~~
- ~~' or 1=1; /*~~
- ~~') or '1'='1~~
- ~~…~~
- Safer

It's good!

ORM seems modern, but ...

the most of ORMs are SLOW.

SQL < ______ < ORM

SQL < MoSQL < ORM

SQL Builders

SQL Builders (cont.)

```
>>> from mosql.build import *

>>> select('pycon')
SELECT * FROM "pycon"

>>> select('pycon', {'id': 'mosky'})
SELECT * FROM "pycon" WHERE "id" = 'mosky'
```

SQL Builders (cont.)

```
>>> insert('pycon', {'yr': 2013, 'id': 'masky'})
INSERT INTO "pycon" ("id", "yr") VALUES ('masky', 2013)

>>> update('pycon',
...     where={'id': 'masky'},
...     set  ={'id': 'mosky'}
... )
UPDATE "pycon" SET "id"='mosky' WHERE "id" = 'masky'
```

SQL Builders (cont.)

- `insert(table, set, …)`
- `select(table, where, …)`
- `update(table, where, set, …)`
- `delete(table, where, …)`
- `...`

If you like it,

```
sudo pip install mosql
```

Model of Result Set

Model: Configure Connection

```
import psycopg2.pool
from mosql.result import Model


pool = psycopg2.pool.SimpleConnectionPool(1, 5, 
database='mosky')


class PostgreSQL(Model):
    getconn = pool.getconn
    putconn = pool.putconn
```

Model: Set the Name of Table

```
class Person(PostgreSQL):
    table = 'person'


>>> Person.select({'person_id': 'mosky'})
{'name': ['Mosky Liu'], 'person_id': ['mosky']}


>>> Person.where(person_id=('andy', 'mosky'))
{'name': ['Andy Warhol', 'Mosky Liu'], 'person_id':
['andy', 'mosky']}
```

Model: Make Queries

```
Person.select({'person_id': 'mosky'})
Person.insert({'person_id': 'tina'})
Person.update(
    where={'person_id': 'mosky'},
    set  ={'name'     : 'Yiyu Liu'}
)
Person.delete({'person_id': 'tina'})
```

Model: Squash Columns

```
class Person(PostgreSQL):
    table    = 'person'
    squashed = set(['person_id', 'name'])

>>> Person.select({'person_id': 'mosky'})
{'name': 'Mosky Liu', 'person_id': 'mosky'}

>>> Person.where(person_id=('andy', 'mosky'))
{'name': 'Andy Warhol', 'person_id': 'andy'}
```

Model: Arrange

```
class Person(PostgreSQL):
    ...
    arrange_by = ('person_id', )

>>> for person in Person.arrange({'person_id':
('andy', 'mosky')}):
...     print person
{'name': 'Andy Warhol', 'person_id': 'andy'}
{'name': 'Mosky Liu', 'person_id': 'mosky'}
```

Model: Arrange (cont.)

```
>>> for detail in Detail.arrange({'person_id':
('mosky', 'andy')}):
...     print detail
...
{'detail_id': [5],
 'key': 'email',
 'person_id': 'andy',
 'val': ['andy@gmail.com']}
...
```

Model: Find

```
class Person(PostgreSQL):
    ...
    arrange_by = ('person_id', )


>>> for person in Person.find(person_id=('andy',
'mosky')):
...     print person
{'name': 'Andy Warhol', 'person_id': 'andy'}
{'name': 'Mosky Liu', 'person_id': 'mosky'}
```

Model: Identify a Row

```
class Person(PostgreSQL):
    ...
    ident_by   = ('person_id', )
```

Model: Modification

```
>>> p = Person.where(person_id='mosky')
>>> p['name'] = 'Yiyu Liu'
>>> p.name = 'Yiyu Liu'
>>> p.save()

>>> d = Detail.where(person_id='mosky', key='email')
>>> p['val'][0] = '<modified email>'
>>> p.val[0] = '<modified email>'
>>> p.save()
```

Model: Pop and Append

```
>>> d = Detail.where(person_id='mosky', key='email')
>>> p.pop(-1)
>>> p.append({'val': '<new mail>'})
>>> p.save()
```

Model: Default Clauses

```
class Person(PostgreSQL):
    ...
    clauses = dict(
        order_by=('person_id', )
    )
```

Performance

- About 4x faster than SQLAlchemy.
- Just a little bit slow than pure SQL.

Security

- Security by default.
- Use escaping technique.
- Prevent SQL injection from both value and identifier.
- Passed the tests from sqlmap at level=5 and risk=3.

Conclusion

- Easy-to-Learn
- Convenient
- Faster
- Secure
- `sudo pip install mosql`
- http://mosql.mosky.tw/
- Welcome to fork!