

# Introduction to python ORM SQLAlchemy

Jorge A. Medina Oliva - Barcelona

<http://pybcn.org>

May. 22 , 2014

## What is an ORM?

This means: maps the tables and columns in a relational database directly to the object instance and wraps all SQL/DDl functionality in his methods.

# SQLAlchemy it's equivalent to...

- Hiberante in java (main ideas become from here)
- Doctrine in PHP
- DataMapper in ruby (this is more close to Active Record pattern)
- NHibernate in .Net C#
- django ORM in python Web framework

## differences with django orm

There are two very important differences between SQLAlchemy and Django. SQLAlchemy is a deeply layered system, whereas Django's ORM is basically just one layer which is the ORM you see.

- 1 In SQLAlchemy you have at the very bottom the engine:
  - Connection pools and basic API differences between different databases
  - On top of that: the SQL abstraction language,
  - On top of SQL abstraction: the table definitions with the basic ORM
  - And on top of ORM you have the declarative ORM which looks very close to the Django ORM.
- 2 The other more striking difference however is that SQLAlchemy follows the “Unit of Work” pattern whereas Django's ORM follows something that is very close to the “Active Record” pattern.

## SQLAlchemy Advantages

- Great documentation at <http://www.sqlalchemy.org>
- Independent framework
- active and stable development
- Strong design since beginning
- Layered components like Connection pool, ORM, API, SQL, Aggregates, etc.
- We can use any independent component
- Implement advanced Hibernate's ideas.
- SQLAlchemy doesn't override all your columns when you just changed one on update.
- differed column load
- Connection pooling

# SQLAlchemy Disadvantages

- Documentation overwhelming
- Confuses at the beginning because exist different programming options.
- only integrates in Flask-SQLAlchemy web framework

## Supported Platforms

**SQLAlchemy has been tested against the following platforms:**

- cPython since version 2.6, through the 2.xx series
- cPython version 3, throughout all 3.xx series
- Pypy 2.1 or greater

# Installation

With pip

```
pip install SQLAlchemy
```

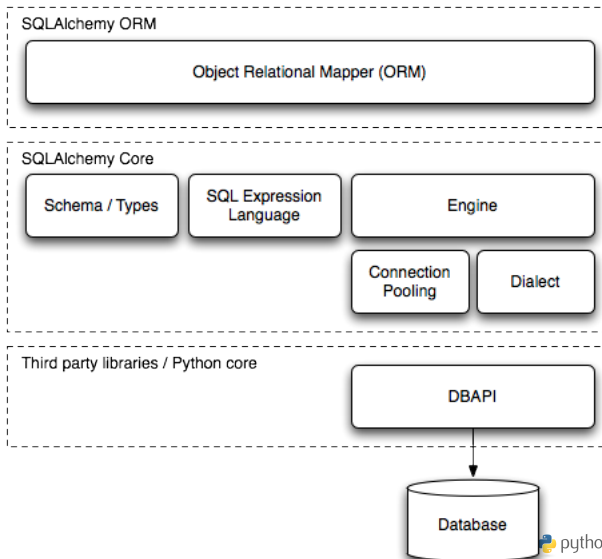
Or with easy\_install

```
easy_install SQLAlchemy
```

*All other methods are supported too...*



# Arquitecture of SQLAlchemy



# The basics to starts

```
""" SQLAlchemy engine factory """
from sqlalchemy import create_engine
""" ORM Datatypes """
from sqlalchemy import Column, ForeignKey, Integer, String
""" ORM Session factory """
from sqlalchemy.orm import sessionmaker
""" ORM relationship mapper """
from sqlalchemy.orm relationship
""" Declarative Base Object """
from sqlalchemy.ext.declarative import declarative_base
""" SQL expresions functions wrappers """
from sqlalchemy.sql import func
```

# Basic Use case

```
engine = create_engine('sqlite:///demo.db', echo=False)
session = sessionmaker(bind=engine)()
Base.metadata.create_all(engine, checkfirst=True)
p = Parent('prueba')
session.add(p)
session.commit()
```

# Basic Use case

```
q = session.query(Parent).all()
for x in q:
    c = Child("child", x.id)
    session.add(c)
session.commit()
session.refresh(p)
for x in q:
    print("{}+\\n      |".format(x.name))
    for i in x.children:
        print("      +-->{}".format(i.name))
```

# Object mapper and relational pattern

## One to Many

```
Base = declarative_base()
```

```
class Parent(Base):
    __tablename__ = 'parent'
    id = Column(Integer, primary_key=True)
    children = relationship("Child", backref="parent")

class Child(Base):
    __tablename__ = 'child'
    id = Column(Integer, primary_key=True)
    parent_id = Column(Integer, ForeignKey('parent.id'))
```

# Object mapper and relational pattern

## Many to One

```
class Parent(Base):
    __tablename__ = 'parent'
    id = Column(Integer, primary_key=True)
    child_id = Column(Integer, ForeignKey('child.id'))
    child = relationship("Child")

class Child(Base):
    __tablename__ = 'child'
    id = Column(Integer, primary_key=True)
```

# Object mapper and relational pattern

## One to One

```
class Parent(Base):
    __tablename__ = 'parent'
    id = Column(Integer, primary_key=True)
    child = relationship("Child", uselist=False,
        backref="parent")

class Child(Base):
    __tablename__ = 'child'
    id = Column(Integer, primary_key=True)
    parent_id = Column(Integer, ForeignKey('parent.id'))
```

# Object mapper and relational pattern

## Many to Many

```
association_table = Table('association', Base.metadata,
    Column('left_id', Integer, ForeignKey('left.id')),
    Column('right_id', Integer, ForeignKey('right.id'))
)
```

```
class Parent(Base):
    __tablename__ = 'left'
    id = Column(Integer, primary_key=True)
    children = relationship("Child",
        secondary=association_table,
        backref="parents")
```

```
class Child(Base):
    __tablename__ = 'right'
    id = Column(Integer, primary_key=True)
```



# Bibliography

- 1 SQLAlchemy - Official documentation, March 28, 2014  
<http://docs.sqlalchemy.org>
- 2 Armin Ronacher - SQLAlchemy and You, July 19, 2011  
<http://lucumr.pocoo.org/2011/7/19/sqlalchemy-and-you/>
- 3 Alexander Solovyov, April 23, 2011  
<http://solovyov.net/en/2011/basic-sqlalchemy/>
- 4 Alexander Solovyov, May 14, 2012  
<https://github.com/piranha/slides/blob/gh-pages/sqla-talk/sqla.md>

# Thanks & Contact info

## Questions?

More information or social links about me:

<http://bsdchile.cl>