

SQLAlchemy and Elixir

Jonathan LaCour

<http://cleverdevil.org>

jonathan@cleverdevil.org

What is SQLAlchemy?

“SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.”

What is Elixir?

“Elixir is a declarative layer on top of SQLAlchemy. It is a thin wrapper, which provides the ability to define model objects following the Active Record design pattern, and using a DSL syntax.”

Plan of Action

- SQLAlchemy Overview
 - Concepts
 - Example
- Elixir Overview
 - Concepts
 - Example
- Questions and Advanced Topics

SQLAlchemy Concepts

- Engine
- MetaData
- Table
- Mapper
- Session
- Query

SQLAlchemy Concepts: Engine

- Connection handling and pooling.
- Defines the “dialect” for database-specific translation.
- PostgreSQL, MySQL, Oracle, MS-SQL, Firebird, Informix, SQLite.

```
# create an engine
engine = create_engine('postgres://user:pass@host/database')

# perform a query
connection = engine.connect()
result = connection.execute('select * from users')
for row in result:
    print row['username'], row['email']

# close the connection
connection.close()
```

SQLAlchemy Concepts: **MetaData**

- A collection of “schema items” like tables and indexes.
- Can be “bound” to an engine.

```
# create a metadata and bind it to an already-created engine  
metadata = MetaData()  
metadata.bind = engine
```

```
# issue CREATE statements for objects within this metadata  
metadata.create_all()
```

SQLAlchemy Concepts: Table

- A table in your SQL database.
- Associated with a MetaData.
- Can perform SQL operations against tables.
- Can be “reflected” from existing schemas (autoload=True).

```
users_table = Table('users', metadata,  
    Column('id', Integer, primary_key=True),  
    Column('username', Unicode(16), nullable=False),  
    Column('email', Unicode(60)),  
    Column('password', Unicode(20), nullable=False)  
)
```

SQLAlchemy Concepts: Table

```
# insert a row into the users table
users_table.insert().execute(
    username='cleverdevil',
    email='jonathan@cleverdevil.org',
    password='s3cr3tw0rd'
)
```

SQLAlchemy Concepts: Table

```
# select all rows from the users table
rows = users_table.select().execute()
for row in rows:
    print row['username'], row['email']

# select certain rows from the users table
clever_not_devil_rows = users_table.select(
    and_(users_table.c.username.like('%clever%'),
         not_(users_table.c.username.like('%devil%')))
).execute()
```

SQLAlchemy Concepts: Mapper

- Part of the Object Relational Mapper (ORM).
- Associates a table (or other “selectable” schema object) with a class.
- Defines relationships between objects.

```
class User(object):  
    def __init__(self, username, email, password):  
        self.username = username  
        self.email = email  
        self.password = password
```

```
mapper(User, users_table)
```

SQLAlchemy Concepts: Session

- Part of the Object Relational Mapper (ORM).
- Represents a “Unit of Work” which is a collection of objects to be persisted.
- Objects are “saved” to a session, then “flushed” to the database.

```
# create a session and a user object
session = create_session()
```

```
user = User()
user.username = 'cleverdevil'
user.email = 'jonathan@cleverdevil.org'
user.password = 's3cr3tw0rd'
```

```
# save the object to the session and flush it
session.save(user)
session.flush()
```

SQLAlchemy Concepts: Query

- Query objects used to fetch mapped objects from the database.
- Often created through a session.
- Generatively build select statements.

```
# create a query object for User class  
query = session.query(User)
```

SQLAlchemy Concepts: Query

```
# get all User objects from the database
for user in query.all():
    print user.name, user.email

# get a single User object from the database
cleverdevil = query.get_by(username='cleverdevil')
```

SQLAlchemy Concepts: Query

```
# get the top 5 users with "clever" but not
# "devil" in their username in alphabetical order
users = query.filter(
    User.c.username.like('%clever%')
).filter(
    ~User.c.username.like('%devil%')
).order_by(User.c.username).limit(5)
```

SQLAlchemy Example

- Weblog Software
- People write Articles
- Plan: Tables, Classes, Mappers

SQLAlchemy Example: Tables

```
metadata = MetaData()
```

```
person_table = Table('person', metadata,  
    Column('id', Integer, primary_key=True),  
    Column('name', Unicode),  
    Column('email', Unicode),  
    Column('password', Unicode)  
)
```

```
article_table = Table('article', metadata,  
    Column('id', Integer, primary_key=True),  
    Column('title', Unicode),  
    Column('description', Unicode),  
    Column('content', Unicode),  
    Column('author_id', Integer, ForeignKey(person_table.c.id))  
)
```

SQLAlchemy Example: Objects

```
class Person(object):
    def __init__(self, name=None, email=None, password=None):
        self.name = name
        self.email = email
        self.password = password
```

```
class Article(object):
    def __init__(self, title=None, description=None,
                 content=None, author=None):
        self.title = title
        self.description = description
        self.content = content
        self.author = author
```

SQLAlchemy Example: Mappers

```
mapper(Person, person_table, properties={  
    'articles' : relation(Article)  
})
```

```
mapper(Article, article_table, properties={  
    'author' : relation(Person)  
})
```

SQLAlchemy Example: Demonstration

```
# bind our metadata to an in-memory
# sqlite database, and create tables
metadata.bind = 'sqlite:/// '
metadata.create_all()

# create a Person instance
jonathan = Person(
    name='Jonathan'
    email='jonathan@cleverdevil.org'
    password='s3cr3t'
)

create an Article instance
blog_post = Article(
    title='Some Blog Post',
    description='Some description',
    content=Some content,
    author=jonathan
)
```

```
# create a session and save the post, which
# will also save the dependant Person object
session = create_session()
session.save(blog_post)
session.flush()

# create another session and use it to query
# for our just created instance
session = create_session()
query = session.query(Person)
jonathan = query.get_by(name='Jonathan')

# walk through the object, printing out some
# properties, including related objects
print jonathan.name
print jonathan.articles[0].title
print jonathan.articles[0].author.name
```

SQLAlchemy: In Summary

- We have just scratched the surface: SQLAlchemy is amazingly powerful.
 - Polymorphic inheritance.
 - Complex relationships.
 - Mapping to arbitrary selectable.
 - Useful extensions.
 - Much, much more!
- Excellent SQL toolkit and connection handling.
- Extremely powerful optional ORM.

SQLAlchemy: Mike Bayer and Chuck Norris

Mike Bayer is a robot built by a higher order of beings who think in relational algebra, and can bend space-time to give him 36-48 hours of productive time in a single day.

He doesn't sleep, he just codes with his eyes closed.

Elixir: Design Patterns

- **SQLAlchemy** uses the “**Data Mapper**” design pattern
 - Classes and tables **aren't** one-to-one.
 - Classes can be mapped to arbitrary selects.
- **Elixir** uses the “**Active Record**” design pattern.
 - Classes and tables **are** one-to-one.
 - Classes, tables, and mappers can be defined in one step.
 - Unlike some other “Active Record” ORMs, compound primary keys **are** supported!

Elixir Concepts: Entity

- Defines your class, table, and mapper in a single step.
- Uses an easy-to-read domain-specific language (DSL).

```
class User(Entity):  
  has_field('username', Unicode(16))  
  has_field('email', Unicode(60))  
  has_field('password', Unicode(20))
```

Elixir Example

- Weblog Software (again) – People write Articles
- Plan: define entities.

```
class Person(Entity):  
  has_field('name', Unicode)  
  has_field('email', Unicode)  
  has_field('password', Unicode)  
  
  has_many('articles', of_kind='Article', inverse='author')
```

```
class Article(Entity):  
  has_field('title', Unicode)  
  has_field('description', Unicode)  
  has_field('content', Unicode)  
  
  belongs_to('author', of_kind='Person', inverse='articles')
```

Elixir Example: Demonstration

```
# bind our metadata to an in-memory
# sqlite database, and create tables
metadata.bind = 'sqlite:/// '
metadata.create_all()

# create a Person instance
jonathan = Person(
  name='Jonathan'
  email='jonathan@cleverdevil.org'
  password='s3cr3t'
)

create an Article instance
blog_post = Article(
  title='Some Blog Post',
  description='Some description',
  content=Some content,
  author=jonathan
)
```

```
# elixir automatically creates a threadlocal
# session for you using "SessionContext"
objectstore.flush(); objectstore.clear()

# elixir provides a convenience function for
# getting query objects
query = Person.query()
jonathan = query.get_by(name='Jonathan')

# walk through the object, printing out some
# properties, including related objects
print jonathan.name
print jonathan.articles[0].title
print jonathan.articles[0].author.name
```

Elixir vs. SQLAlchemy

- Lines of Code for Weblog Data Model
 - **SQLAlchemy** weighs in at **38 lines**.
 - **Elixir** comes in at **16 lines**.
 - That's 58% less code.
- Sometimes you don't need all the power of SQLAlchemy
- But Elixir is pretty powerful in its own right....

More Elixir: Events

- Watch for changes on your entities
- Decorators for before and after insert, update, and delete.

```
from elixir import *
from elixir.events import after_update

class Article(Entity):
    has_field('title', Unicode)
    has_field('description', Unicode)
    has_field('content', Unicode)
    belongs_to('author', of_kind='Person', inverse='articles')

@after_update
def notify_author(self):
    print 'Notifying', self.author.email
    print ' -> article:', self.title, 'was updated'
```

More Elixir: Encryption

- Automatically encrypts and decrypts fields using Blowfish.

```
from elixir import *
from elixir.ext.encrypted import acts_as_encrypted

class Person(Entity):
  has_field('name', Unicode)
  has_field('email', Unicode)
  has_field('password', Unicode)

  has_many('articles', of_kind='Article', inverse='author')
  acts_as_encrypted(for_fields=['password'], with_secret='abc123')
```

More Elixir: Versioning

- Automatically tracks versions of your entities.
- Reverting, version comparison, timestamps, version numbers, after_revert event.

```
from elixir import *
from elixir.ext.versioned import *

class Article(Entity):
    has_field('title', Unicode)
    has_field('description', Unicode)
    has_field('content', Unicode)
    belongs_to('author', of_kind='Person', inverse='articles')
    acts_as_versioned()

    @after_revert
    def notify_author(self):
        print 'Notifying', self.author.email
        print ' -> article:', self.title, 'was reverted'
```

More Elixir: Versioning

```
# loop through previous versions of an article, showing differences
article = Article.query().get_by(title='Some Post')
print 'Current version:', article.version, article.timestamp
for version in article.versions:
    print ' * Old version:', version.version, version.timestamp
    print ' - differences ->', article.compare_with(version)

# get the version of this article as of a date and time
version = article.get_as_of(timestamp)

# revert to previous version
article.revert()

# revert to specific version number
article.revert_to(2)
```

More Elixir: [Associables](#)

- Create common association patterns easily
- Examples: addressable, taggable.

```
from elixir.ext.associable import *

class Tag(Entity):
    has_field('name', Unicode)

acts_as_taggable = associable(Tag)

class Article(Entity):
    has_field('title', Unicode)
    has_field('description', Unicode)
    has_field('content', Unicode)

    acts_as_taggable('tags')
```

```
# loop through the tags on an article
for tag in article.tags:
    print 'Article', article.title
    print ' -> tagged as', tag.name

# fetch articles that are tagged with a
# certain tag
python_posts = Article.select_by_tags(
    name='python'
)
```

More Elixir: Lower Level Operations

- Elixir is built on SQLAlchemy.
- You can always drop down to table objects.

```
results = select(
    [Person.c.email, Article.c.title],
    and_(
        Person.c.name == 'Jonathan',
        Person.c.id==Article.c.author_id
    )
).execute()

person_rows = Person.table.select().execute()
```

Elixir: Summary

- Simplified layer on SQLAlchemy
 - Active Record instead of Data Mapper
 - Lower-level operations always available
- Cross-cutting functionality through Elixir Statements
 - Associable
 - Encryption
 - Versioning
 - Write your own!

Questions

```
can_i_do_it_with_sqlalchemy = lambda it: 'Yep'
```

[Elixir](http://elixir.ematia.de) <http://elixir.ematia.de>

[SQLAlchemy](http://sqlalchemy.org) <http://sqlalchemy.org>

[Jonathan LaCour](http://cleverdevil.org) <http://cleverdevil.org>