Redpill Linpro

Introduction to SQLAlchemy

SQLAlchemy

python

The Python SQL Toolkit and Object Relational Mapper

1 What is SQLAlchemy?

What is SQLAlchemy?

SQLAlchemy is a Python Library created by Mike Bayer to provide a high-level Pythonic interface to relational databases such as *Oracle*, *MySQL*, *PostgreSQL*, *DB2* and *SQLite*.

SQLAlchemy includes a database server-independent SQL expression language and an object-relational mapper (ORM).

2 Why use SQLAlchemy?

Why use SQLAlchemy?

- portability - the programming interface is independent of the type of database-server that is used
- security - no more SQL injections
- abstraction - no need to bother with complex JOINs
- object-orientation - you work with objects and *NOT* tables and rows
- performance - exploits the likehood of reusing a particular query
- flexibility - you can override almost anything

3 Basic architecture

Basic architecture

SQLAlchemy consists of several components, including the *SQL expression language* and the *ORM*.

In order to enable these components, *SQLAlchemy* also provides an `Engine` class and `MetaData` class.

- Engine - manages the *SQLAlchemy* connection pool and the database-independent SQL dialect layer
- MetaData - used to collect and organize information about your table layout (database schema)
- SQL expression language - provides an API to execute your queries and updates against your tables, all from Python, and all in a database-independent way
- ORM - provides a convenient way to add database persistence to your Python objects without requiring you to design your objects around the database, or the database around the objects.

SQLAlchemy uses the *data mapper* pattern.

4 Example

Example

```
psql -U postgres
```

We initialize a PostgreSQL user and database:

```
CREATE USER redlabxuser WITH PASSWORD 'redlabxpasswd';
CREATE DATABASE redlabxdb WITH OWNER = redlabxuser;
```

We create an `Engine` object

```
>>> import sqlalchemy
>>> sqlalchemy.__version__

>>> from sqlalchemy import create_engine

# >>> engine = create_engine('postgresql://redlabxuser:redlabxpasswd@localhost/redlabxdb' , echo=True)
# The string form of the URL is dialect+driver://user:password@host/dbname[?key=value..],
# where dialect is a database name such as mysql, oracle, postgresql, etc.,
# and driver the name of a DBAPI, such as psycopg2, pyodbc, cx_oracle,

# The echo flag is a shortcut to setting up SQLAlchemy logging,
# which is accomplished via Python's standard logging module.
# With it enabled, we'll see all the generated SQL produced.
>>> engine = create_engine('sqlite:///library.db', echo=True)
```

5 Example

Table definition / Classical mapping

Next we are ready to define and create our tables

```
>>> from sqlalchemy import Table, Column, Integer, String, MetaData, ForeignKey, text

>>> metadata = MetaData()

>>> authors_table = Table('authors', metadata,
Column('id', Integer, primary_key=True), Column('name', String))
>>> books_table = Table('books', metadata,
        Column('id', Integer, primary_key=True),
        Column('title', String),
        Column('description', String),
        Column('author_id', ForeignKey('authors.id')))
# Column('name', String(50)) is possible

>>> metadata.create_all(engine) # creates the tables
```

6 Example

Using the Table objects directly

We don't even need to map our Table-objects in order to manipulate our tables

```
>>> insert_stmt = authors_table.insert(bind=engine)
>>> type(insert_stmt)
<class 'sqlalchemy.sql.expression.Insert'>
>>> print (insert_stmt)
INSERT INTO authors (id, name) VALUES (:id, :name)

>>> compiled_stmt = insert_stmt.compile()
>>> print(compiled_stmt.params)
{'id': None, 'name': None}

>>> insert_stmt.execute(name='Alexandre Dumas')  # insert a single entry
>>> insert_stmt.execute([{'name': 'Mr X'}, {'name': 'Mr Y'}]) # a list of entries

>>> metadata.bind = engine  # no need to explicitly bind the engine from now on
>>> select_stmt = authors_table.select(authors_table.c.id==2)
>>> result = select_stmt.execute()
>>> result.fetchall()
[(1, u'Mr X')]

>>> del_stmt = authors_table.delete()
>>> del_stmt.execute(whereclause=text("name='Mr Y'"))
>>> del_stmt.execute()  # delete all
```

7 Example

Classical mapping

Now we can define our classes and create a mapping

```
>>> from sqlalchemy.orm import mapper
>>> from sqlalchemy.orm import relationship, backref

>>> class Author(object):
...     def __init__(self, name):
...         self.name = name
...
...     def __repr__(self):
...         return self.name

>>> class Book(object):
...     def __init__(self, title, description, author):
...         self.title = title
...         self.description = description
...         self.author = author
...
...     def __repr__(self):
...             return self.title

>>> mapper(Book, books_table)
>>> mapper(Author, authors_table, properties = {
...     'books': relation(Book, backref='author')})
```

8 Example

Declarative mapping

Doing the same thing the easy way:

```
>>> from sqlalchemy.ext.declarative import declarative_base
>>> from sqlalchemy.orm import relationship, backref

>>> Base = declarative_base()
>>> class Author(Base):
...     __tablename__ = 'authors'
...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...
...     def __init__(self, name):
...         self.name = name
...
...     def __repr__(self):
...         return self.name
```

9 Example

Declarative mapping

```
>>> class Book(Base):
...     __tablename__ = 'books'
...     id = Column(Integer, primary_key=True)
...     title = Column(String)
...     description = Column(String)
...     author_id = Column(Integer, ForeignKey('authors.id'))
...     author = relationship(Author, backref=backref('books', order_by=title))
...
...     def __init__(self, title, description, author):
...         self.title = title
...         self.description = description
...         self.author = author
...
...     def __repr__(self):
...         return self.title

>>> Base.metadata.create_all(engine) # create tables
```

10 Example

Creating instances

```
>>> from sqlalchemy.orm import sessionmaker
>>> Session = sessionmaker(bind=engine) # bound session

>>> session = Session()

>>> author_1 = Author('Richard Dawkins')
>>> author_2 = Author('Matt Ridley')
>>> book_1 = Book('The Red Queen', 'A popular science book', author_2)
>>> book_2 = Book('The Selfish Gene', 'A popular science book', author_1)
>>> book_3 = Book('The Blind Watchmaker', 'The theory of evolutio', author_1)

>>> session.add(author_1)
>>> session.add(author_2)
>>> session.add(book_1)
>>> session.add(book_2)
>>> session.add(book_3)
# or simply  session.add_all([author_1, author_2, book_1, book_2, book_3])

>>> session.commit()

>>> book_3.description = u'The theory of evolution' # update the object
>>> book_3 in session # check whether the object is in the session
True
>>> session.commit()
```

11 Example

Querying

```
>>> session.query(Book).order_by(Book.id) # returns a query
>>> session.query(Book).order_by(Book.id).all() # returns an object-list

# return all book objects where title == 'The Selfish Gene'
>>> session.query(Book).filter(Book.title == 'The Selfish Gene').order_by(Book.id).all()

# using LIKE
>>> session.query(Book).filter(Book.title.like('The%')).order_by(Book.id).all()

>>> query = session.query(Book).filter(Book.id == 9).order_by(Book.id)
>>> query.count() # returns 0L
>>> query.all() # returns an empty list
>>> query.first() # returns None
>>> query.one() # raises NoResultFound exception

>>> query = session.query(Book).filter(Book.id == 1).order_by(Book.id)
>>> book_1 = query.one()
>>> book_1.description # returns u'A popular science book'
>>> book_1.author.books # returns a list of Book-objects representing all the books from the same author.

# get a list of all Book-instances where the author's name is 'Richard Dawkins'
>>> session.query(Book).filter(Book.author_id==Author.id).filter(Author.name=='Richard Dawkins').all()
>>> session.query(Book).join(Author).filter(Author.name=='Richard Dawkins').all()
>>> session.query(Book).\
...     from_statement('SELECT b.* FROM books b, authors a WHERE b.author_id = a.id AND a.name=:name').\
...     params(name='Richard Dawkins').all()
```