ABOUT ME

• Software & Data
• Python since 2001 ( 1.5.2 )
• Business Intelligence
• Data Extraction & Engineering
• Data Analysis & Data Science
TABLE OF CONTENTS

1. Developing for RDBMS
2. SQLAlchemy Core
3. SQLAlchemy ORM
CREATE TABLE groups (  
id serial primary key,  
name character varying,  
begin_year smallint,  
end_year smallint  
);  

class Group:  
    def __init__(self, name, begin_year, end_year):  
        self.name = name  
        self.begin_year = begin_year  
        self.end_year = end_year
TABLE-CLASS MAPPING

```
insert into groups (name, begin_year, end_year)
values ("Metallica", 1981, null);

postgres=> select * from groups where name = 'Metallica';

id |   name    | begin_year | end_year
----+-----------+------------+----------
 93 | Metallica |       1981 |          
(1 row)

>>> metallica = Group ("Metallica", 1981, None)
>>> metallica.name
'Metallica'
```
SQL STRINGS SEEM EASY

```python
>>> name = "Robert'\); DROP TABLE students;--"
>>> "select * from students where ( name = '{})\);" .format(name)
select * from students where ( name = 'Robert'); DROP TABLE students;--' );
```
DATA ACCESS LAYER WISH-LIST ( FOR RDBMS )

- Wrap DB actions in Idiomatic API.
- Reduce boilerplate when possible.
- Safety & Correctness.
- Support multiple backends.
"THE PYTHON SQL TOOLKIT AND OBJECT RELATIONAL MAPPER"

AUTHOR: MICHAEL BAYER (@ZZZEEK)

RELEASE: 2005
STRUCTURE OF SQLALCHEMY

- **SQLAlchemy ORM**
  - Object Relational Mapper (ORM)

- **SQLAlchemy Core**
  - Schema / Types
  - SQL Expression Language
  - Engine
  - Connection Pooling
  - Dialect

- **Third party libraries / Python core**
  - DBAPI

- Database
AN EXISTING (POSTGRESQL) DATABASE

(adapted from musicbrainz.org)
WHERE DO WE START?

CONNECT TO A DATABASE

```python
from sqlalchemy import create_engine

engine = create_engine("postgres://musicbrainz:musicbrainz@localhost/musicbrainz")

with engine.connect() as conn:
    ...
    rp = conn.execute("select * from musicians where name ~ 'D...d.B..ie'")
    print(rp.fetchall())
```

DO SOMETHING

```python
[(810693, 'David Bodie'), (956, 'David Bowie')]
```
MANUALLY DECLARING THE METADATA

going to manually declare the metadata using the `MetaData` class from SQLAlchemy.

```python
from sqlalchemy import MetaData
from sqlalchemy import Table, Column, ForeignKey, Index
from sqlalchemy import INTEGER, SMALLINT, VARCHAR

metadata = MetaData()

musicians = Table('musicians', metadata,
                  Column('id', INTEGER(), primary_key=True),
                  Column('name', VARCHAR())
)

groups = Table('groups', metadata,
               Column('id', INTEGER(), primary_key=True),
               Column('name', VARCHAR()),
               Column('location', VARCHAR()),
               Column('begin_year', SMALLINT()),
               Column('end_year', SMALLINT())
)

musician_in_group = Table('musician_in_group', metadata,
                           Column('musician_id', INTEGER(), ForeignKey('musicians.id')),
                           Column('group_id', INTEGER(), ForeignKey('groups.id')),
                           Column('begin_year', SMALLINT()),
                           Column('end_year', SMALLINT())
)

metadata.create_all(bind=engine)
```
```python
>>> from sqlalchemy import MetaData
>>> metadata = MetaData()
>>> metadata.reflect(
...     bind = engine,
...     only = [ 'musicians', 'groups', 'musician_in_group' ]
... )
>>> musicians = metadata.tables['musicians']
>>> groups = metadata.tables['groups']
>>> musician_in_group = metadata.tables['musician_in_group']
>>> musicians
Table('musicians', MetaData(bind=None), Column('id', INTEGER()),
```
>> from sqlalchemy import insert

>>> ins = insert(musicians).values( name = "Moran Cowen" )
>>> conn.execute(ins)
<sqlalchemy.engine.result.ResultProxy object at 0x7f100a1ff278>
>>> rp = _
>>> (rp.rowcount, rp.is_insert, rp.inserted_primary_key)
(1, True, [1517250])

>>> ins
<sqlalchemy.sql.dml.Insert object at 0x7f2a257ea940>

>>> compiled = ins.compile( dialect = engine.dialect )

>>> str(compiled)
'INSERT INTO musicians (name) VALUES (%(name)s) RETURNING musicians.id'

>>> compiled.params
{'name': 'Moran Cowen'}
```python
>>> from sqlalchemy import update, delete
>>> upd = (update(musicians).values(name = "Moran Cohen")
...     .where(musicians.c.name == "Moran Cowen"))
>>> rp = conn.execute(upd)
>>> rp.rowcount
1

>>> del_ = delete(musicians).where(musicians.c.name == "Moran Cohen")
>>> rp = conn.execute(del_)
>>> rp.rowcount
1
```
```python
>>> from sqlalchemy import select
>>> q = select([musicians])
>>> print(q)
SELECT musicians.id, musicians.name
FROM musicians

>>> q1 = q.where( musicians.c.name == 'Bob Dylan' )
>>> print(q1)
SELECT musicians.id, musicians.name
FROM musicians
WHERE musicians.name = :name_1

>>> musicians.c.name == 'Bob Dylan'
<sqlalchemy.sql.elements.BinaryExpression object at 0x7f7572803710>

>>> q2 = q.where( musicians.c.name.like( 'Bob D_1%n' ) )
>>> print(q2)
SELECT musicians.id, musicians.name
FROM musicians
WHERE musicians.name LIKE :name_1

>>> conn.execute(q2).fetchall()
[(344913, 'Bob Dalton'), (1516844, 'Bob Dolin'), (17, 'Bob Dylan')]

>>> q3 = q.where( musicians.c.name.op('~')('Bob Dy' )
>>> print(q3)
SELECT musicians.id, musicians.name
FROM musicians
WHERE musicians.name ~ :name_1
```
SELECT - BOOLEAN COMBINATIONS

```python
>>> iron = groups.c.name.op('~*')('iron')
>>> maiden = groups.c.name.op('~*')('maiden')
>>> q = select([ groups.c.name ]).where( iron & maiden )
>>> print(q)
SELECT groups.name
FROM groups
WHERE (groups.name ~* :name_1) AND (groups.name ~* :name_2)
>>> conn.execute(q).fetchall()
[('The Iron Maidens',), ('Iron Maiden',), ('Iron Maiden',)]

>>> the = groups.c.name.op('~')('The')
>>> q2 = select([ groups.c.id, groups.c.name ]).where( iron & maiden & ~ the )
>>> q2 = select([ groups.c.id, groups.c.name ]).where( iron & maiden ).where( ~ the )
>>> conn.execute(q2).fetchall()
[(10042, 'Iron Maiden'), (413736, 'Iron Maiden')]
```
from sqlalchemy import func, select
q = select([func.count(1)], from_obj = [groups])
print(q)
SELECT count(:param_1) AS count_1 
FROM groups
conn.execute(q).fetchall()
[(289186,)]

first_name_re = r'^[^a-zA-Z]*([a-zA-Z]+)[^a-zA-Z]*.*$

grouped_column = func.regexp_replace(musicians.c.name, first_name_re, r'\1')
print(grouped_column)
regexp_replace(musicians.name, :regexp_replace_1, :regexp_replace_2)

count_column = func.count(1)
q = (select([count_column,grouped_column], from_obj = [musicians]) 
  .group_by(grouped_column) 
  .order_by(count_column.desc()))
print(q)
SELECT count(:param_1) AS count_1, regexp_replace(musicians.name, :regexp_replace_2, :regexp_replace_3) AS regexp_replace_1 
FROM musicians GROUP BY regexp_replace(musicians.name, :regexp_replace_2, :regexp_replace_3) ORDER BY count(:param_1) DESC
conn.execute(q.limit(6)).fetchall()
[(5877, 'John'), (5198, 'DJ'), (4970, 'David'), (4334, 'Michael'), (3821, 'Paul')]
SELECT - JOINS, ALIASES, LABELS

```python
>>> q = (select([groups.alias('g')
...               .join( musician_in_group.alias( 'mig') )
...               .join( musicians.alias( 'm') )])
...)

>>> print(q)
SELECT g.id, g.name, g.location, g.begin_year, g.end_year, mig.musician_id, mig.group_id, mig.begin_year, mig.end_year, m.id, m.name
FROM groups AS g JOIN musician_in_group AS mig ON g.id = mig.group_id JOIN musicians AS m ON m.id = mig.musician_id

>>> q = (select([groups.alias('g')
...               .join( musician_in_group.alias( 'mig') )
...               .join( musicians.alias( 'm') )], use_labels = True)
...)

>>> print(q)
SELECT g.id AS g_id, g.name AS g_name, g.location AS g_location, g.begin_year AS g_begin_year, g.end_year AS g_end_year, mig.musician_id AS mig_musician_id, mig.group_id AS mig_group_id, mig.begin_year AS mig_begin_year, mig.end_year AS mig_end_year, m.id AS m_id, m.name AS m_name
FROM groups AS g JOIN musician_in_group AS mig ON g.id = mig.group_id JOIN musicians AS m ON m.id = mig.musician_id
```
LEFTOVERS

- Window functions
- Recursive CTE’s
SHOW ME THE MAPPING !
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String

Base = declarative_base()

class Musician(Base):
    __tablename__ = "musicians"
    id = Column(Integer, primary_key = True)
    name = Column(String(256))

class Group(Base):
    __tablename__ = "groups"
    id = Column(Integer, primary_key = True)
    name = Column(String(256))
    begin_year = Column(Integer)
    end_year = Column(Integer)

moran = Musician( name = "Moran Cohen" )
my_band = Group( name = "My Band", begin_year = 2017, end_year = None )
WORKING WITH THE SESSION

```python
>>> from sqlalchemy.orm import sessionmaker
>>> Session = sessionmaker( bind = engine )
>>> session = Session()
>>> session.query(Musician).filter(Musician.name == 'Ian Gillan').all()
[<__main__.Musician object at 0x7f2a2603fc88>]
>>> ian = session.query(Musician).filter(Musician.name == 'Ian Gillan').one()
>>> ian
<__main__.Musician object at 0x7f2a2603fc88>
```
PERSISTING NEW OBJECTS

moran = Musician(name = "Moran Cowen")
session.add(moran)
session.commit()
elton = session.query(Musician).filter(Musician.name == 'Elton John').one()
elton.name = "Sir Elton John"
session.flush()
session.commit()
engine = create_engine("postgres://....", echo = True)

import logging

logging.basicConfig(level = logging.INFO)

Session = sessionmaker( bind = engine )

session = Session()

moran = Musician(name = "Moran Cowen")

session.add(moran)

session.flush()
>>> moran.name = "Moran Cohen"
>>> session.flush()
INFO:sqlalchemy.engine.base.Engine:UPDATE musicians SET name=%(name)s WHERE musicians.id = %(musicians_id)s
INFO:sqlalchemy.engine.base.Engine:{ 'name': 'Moran Cohen', 'musicians_id': 15172}
>>> session.commit()
INFO:sqlalchemy.engine.base.Engine:COMMIT

>>> session.delete(moran)
>>> session.commit()
INFO:sqlalchemy.engine.base.Engine:BEGIN (implicit)
FROM musicians
WHERE musicians.id = %(param_1)s
INFO:sqlalchemy.engine.base.Engine:SELECT musicians.id AS musicians_id, musicians.name AS musicians_name
FROM musicians
WHERE musicians.id = %(param_1)s
INFO:sqlalchemy.engine.base.Engine:{ 'param_1': 1517254}
INFO:sqlalchemy.engine.base.Engine:DELETE FROM musicians WHERE musicians.id = %(id)s
INFO:sqlalchemy.engine.base.Engine:{ 'id': 1517254}
INFO:sqlalchemy.engine.base.Engine:COMMIT
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy import Column, Integer, String, ForeignKey

Base = declarative_base()

class MusicianInGroup(Base):
    __tablename__ = "musician_in_group"

    musician_id = Column(Integer, ForeignKey("musicians.id"), primary_key=True)
    group_id = Column(Integer, ForeignKey("groups.id"), primary_key=True)
    begin_year = Column(Integer, primary_key=True)
    end_year = Column(Integer, primary_key=True)

    musician = relationship("Musician")
    group = relationship("Group")

class Musician(Base):
    __tablename__ = "musicians"

    id = Column(Integer, primary_key=True)
    name = Column(String(256))

    groups_assoc = relationship("MusicianInGroup")

class Group(Base):
    __tablename__ = "groups"

    id = Column(Integer, primary_key=True)
    name = Column(String(256))
    begin_year = Column(Integer)
    end_year = Column(Integer)

    musician_assoc = relationship("MusicianInGroup")
LEFTOVERS

• Dialect and Engine customization (Core)
  ▪ Connection pool mgmt
  ▪ Custom SQL Types
• Mapper customization (ORM)
  ▪ Relationship types
  ▪ Inheritance
  ▪ Eager loading
INFLUENCES ON SQLA ORM

- Hibernate (Java)
- SQLObject (Python)
- P of EAA
ORM OR CORE?

ORM

- Complex application-side logic
- Transactions
- (Business Logic, APIs, web)

CORE

- Large insert / delete / update
- Complex queries
- (Analysis, ELT, denormalization)
REFERENCES

- sqlalchemy.org
- Talk Python Podcast, Episode #5: SQLAlchemy and data access in Python
- Fowler's P of EAA
THANK YOU

moran@datafullstack.com

linkedin.com/in/moranski
QUESTIONS ?