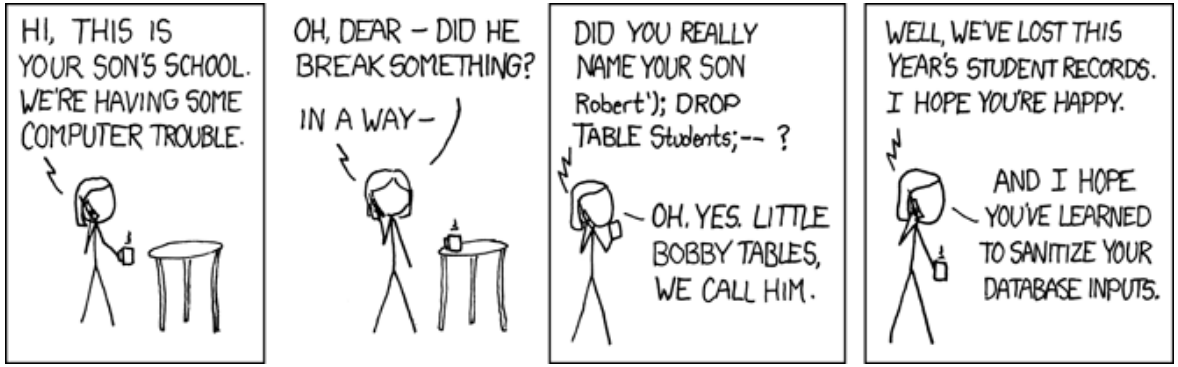

Write Python instead of SQL!

An introduction to SQLAlchemy.

Plain DBAPI+SQL

```
>>> import psycopg2
>>> conn = psycopg2.connect('dbname=test')
>>> cur = conn.cursor()
>>> cur.execute("CREATE TABLE test (id serial PRIMARY KEY, data varchar)")
>>> cur.execute("INSERT INTO test VALUES (%s)", ("Little Bobby 'tables'",))
>>> cur.execute("INSERT INTO test VALUES ('%s')" % ("Little Bobby 'tables'",))
ProgrammingError: syntax error at or near "tables"
LINE 1: INSERT INTO test (data) VALUES ('Little Bobby 'tables'')
```

SQLAlchemy (low-level)

```
>>> from sqlalchemy import *
>>> metadata = MetaData()
>>> table = Table('test', metadata,
                  Column('id', Integer, primary_key=True),
                  Column('name', String))
>>> engine = create_engine('postgresql:///test')
>>> metadata.create_all(engine)
>>> conn = engine.connect()
>>> table.insert(bind=engine).values(name="Little Bobby 'tables'").execute()
>>> conn.execute(select([table])).fetchone()
(1, u"Little Bobby 'tables'")
```

SQLAlchemy (ORM)

```
>>> from sqlalchemy import *
>>> from sqlalchemy.ext.declarative import declarative_base
>>> Base = declarative_base()
>>> class User(Base):
        __tablename__ = 'users'
        id = Column(Integer, primary_key=True)
        name = Column(String)
        def __repr__(self):
            return '<User({}, {})>'.format(self.id, self.name)
>>> engine = create_engine('postgresql:///test')
>>> Base.metadata.create_all(engine)
```

SQLAlchemy (ORM)

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

>>> user = User(name='Bobby Tables')
>>> session.add(user)
>>> session.commit()

>>> session.query(User).one()
<User(1, Bobby Tables)>

>>> user.name = "DROP ALL THE 'TABLES!"
>>> session.query(User).one()
<User(1, DROP ALL THE 'TABLES!)>
>>> session.rollback()
```

Let’s have a relationship

```
>>> class Group(Base):
        __tablename__ = 'groups'
        id = Column(Integer, primary_key=True)
        name = Column(String)
        def __repr__(self):
            return '<Group({}, {})>'.format(self.id, self.name)

>>> class User(Base):
        # ...
        group_id = Column(Integer, ForeignKey('groups.id'), index=True)
        group = relationship('Group', lazy=False, backref='users')
        def __repr__(self):
            return '<User({}, {}, {})>'.format(self.id, self.group, self.name)
```

Let’s have a relationship

```
>>> u = User(name='Bobby Tables')
>>> u2 = User(name='Vimmy McVimface')
>>> g = Group(name='Scriptkiddies')
>>> session.add_all([u, u2, g])
>>> session.commit()

>>> u
<User(1, None, Bobby Tables)>
>>> u2
<User(2, None, Vimmy McVimface)>
>>> g
<Group(1, Scriptkiddies)>
```

Let’s have a relationship

```
>>> u.group = g
>>> g.users
[<User(1, <Group(1, Scriptkiddies)>, Bobby Tables)>]

>>> g.users.append(u2)
>>> u2
<User(2, <Group(1, Scriptkiddies)>, Vimmy McVimface)>
>>> g.users
[<User(1, <Group(1, Scriptkiddies)>, Bobby Tables)>,
 <User(2, <Group(1, Scriptkiddies)>, Vimmy McVimface)>]
```

What else is there

Query options to control column/relationship loading

Cascading along relationships

Custom types

Signals

Why use SQLAlchemy again?

- Sacrifice a little bit of performance / SQL purity for much faster development
- Cleaner / shorter code
- Basically no SQL injection risks
- Excellent documentation http://docs.sqlalchemy.org/en/rel_1_0/
- Very helpful community (#sqlalchemy on freenode)