

# Quick Overview of SQLAlchemy

June 15, 2011 - Boston Python Meetup

Michael Kowalchik

CTO - Smarterer



smarterer

# Who am I?

- Enigneer, programmer, lover of Boston
- CTO and co-founder of Smarterer
- this is my second company, first was: Grazr.  
5 sec history: 3+ years, great tech, great team, no market, painful (but deep) lesson
  - aside: I'll never, ever... write a big project in Perl again.

Smarterer: <http://smarterer.com>

For those that are curious Grazr: <http://techcrunch.com/2006/09/18/grazr-10-blasts-off-into-the-future-of-rss/>

# caveats, privisos, etc...

- SQLAlchemy is big - this is a 10K foot view
- I'm *not* an expert, but I've been messing with it for a while now so YMMV
- Smarterer uses it as the basis of our ORM / database lib
- Speaking of... we launched Monday so running low on sleep :)

# Quick Poll

- I assume everyone knows what an ORM is?
- Have experience with Rails Active Record or Django ORM?
- Have experience with SA?

Previous company leveraged the features of the database heavily. Lots of MySQL veterans on the team. UDF's, Triggers, mysql replication, multiple MySQL storage engines, etc... My history coming to SA. Django very constraining. Didn't like ORMs, too opaque.

# DONT NEED NO NEWFANGLED “ORM”!

I came to SQLAlchemy by “accident”. Originally my dissatisfaction with the philosophy of ORM’s led me to intend to create a db library “closer to the metal”.

My feeling was similar to ORM=vietnam of CS: <http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx>

Started working with SQLAlchemy and realized that was one of it’s main philosophical points. <http://techspot.zzzeek.org/2011/06/16/orm-thoughts-in-140-x-5-characters/>

# Anti ORM?

- Used to using a lot of functions on the database (udfs, replication, triggers, etc...)
- Most ORM's fail to deliver their promised level of abstraction
- Most ORM's marketing: "You'll never have to write SQL! Well... until you have to, so here's an afterthought raw SQL interface"

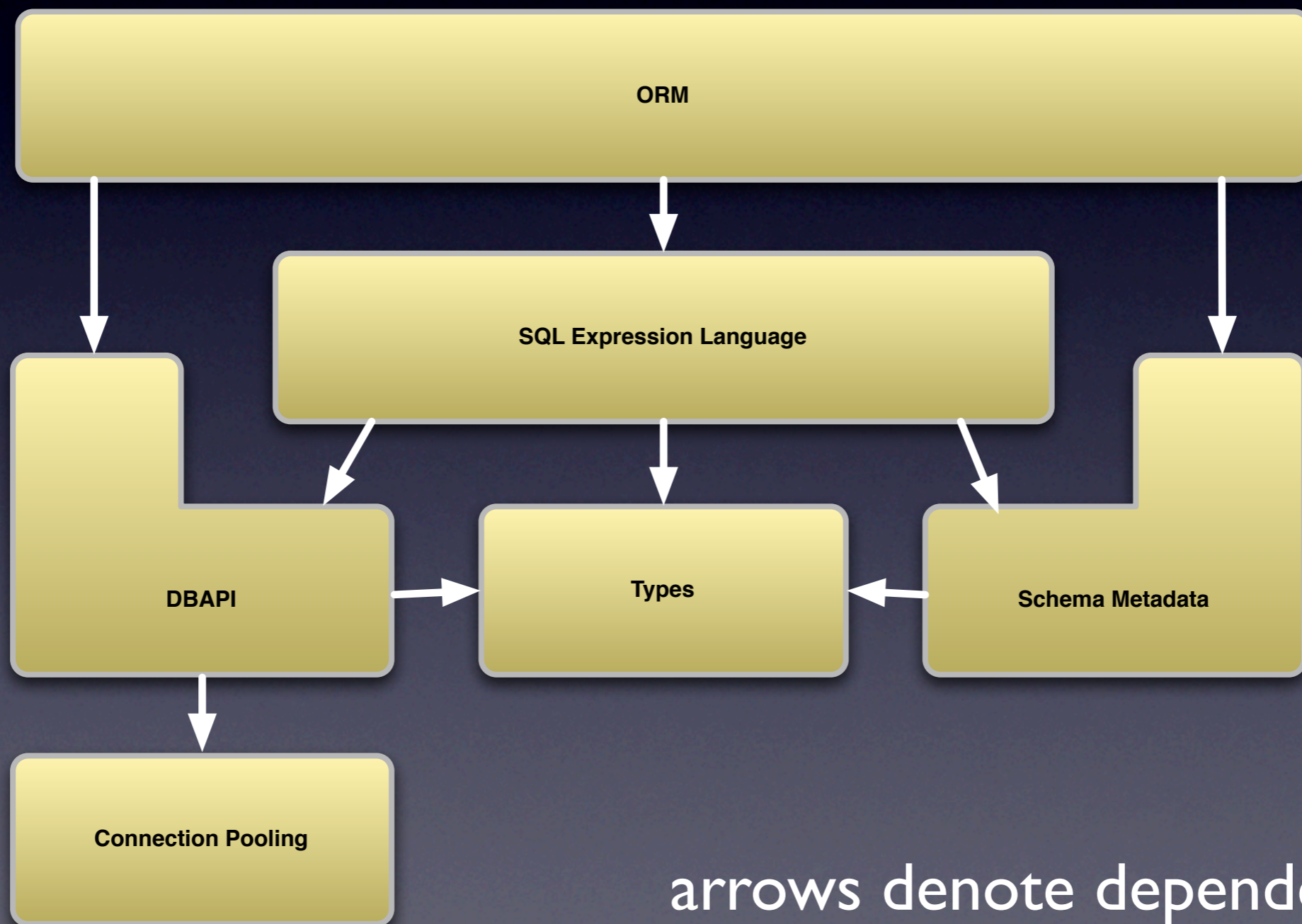
# Anti ORM?

- Don't protect me from the database
- Don't force me not to use the power and features of the database (otherwise why are we using an RDS?)
- Make my life easier, but don't coddle me

# SQLAlchemy, what is it?

- A Python database “toolkit” that includes powerful programmatic SQL expression generation, database abstraction through dialects, **as well as** a powerful ORM

# SQLAlchemy Architecture



Basically a diagram lifted from the SQLAlchemy docs. Shows which elements depend on which other elements.

# Philosophy (my take)

- SQLAlchemy is for: someone who *likes* relational databases, understands their power, likes Python and wants to keep the power of the database
- Not strictly database agnostic - doesn't force a lowest common denominator
- For people who like more flexibility

# SA Concepts

- engine (connection string, dbapi, dialects)
- connection (pools)
- metadata (sa representation of schema)
- expressions - insert, select, update
- ORM, session and Identity Map

# engine

```
from sqlalchemy import create_engine
# connection strings are RFC-1738 style urls
# DIALECT+DRIVER://USERNAME:PASSWORD@HOST:PORT/DATABASE

engine = create_engine('sqlite:///memory:')
```

- engine is SQLAlchemy's representation of the database and its dialect
- uses connection string URLs

# Dialects

```
from sqlalchemy import create_engine
# connection strings are RFC-1738 style urls
# DIALECT+DRIVER://USERNAME:PASSWORD@HOST:PORT/DATABASE
# engine1 = create_engine('mysql://sa_test:@localhost/test')
# engine2 = create_engine('sqlite:///test_db.sqlite')

engine = create_engine('sqlite:///memory:')
```

- dialect is the conversion from SQLAlchemy's internal database representation and the specific database (MySQL, Postgres, Oracle, sqlite, etc...)
- specified in the url

# Connection Pools

- Fairly standard database API pattern
- database connections are expensive operations
- connection pools “hang on” to created connections and re-use them
- request a **new** connection and the pool tries to **recycle** an **existing** connection

# metadata

```
# the object that stores sqlalchemy's understanding of the schema
metadata = MetaData()

# traditional way to define tables
addresses_table = Table('addresses', metadata,
    Column('id', Integer, primary_key=True),
    Column('street', String(255)),
    Column('city', String(255)),
    Column('user_id', Integer, ForeignKey("users.id"), nullable=False)
)
```

- the internal SQLAlchemy representation of the schema.
- Includes tables, columns, relationships

# Expression Language

```
conn = engine.connect()
select_query = select([users_table, addresses_table],
                      (users_table.c.id==addresses_table.c.id) & (users_table.c.name=="bob"))

result = conn.execute(select_query)
for row in result:
    print row

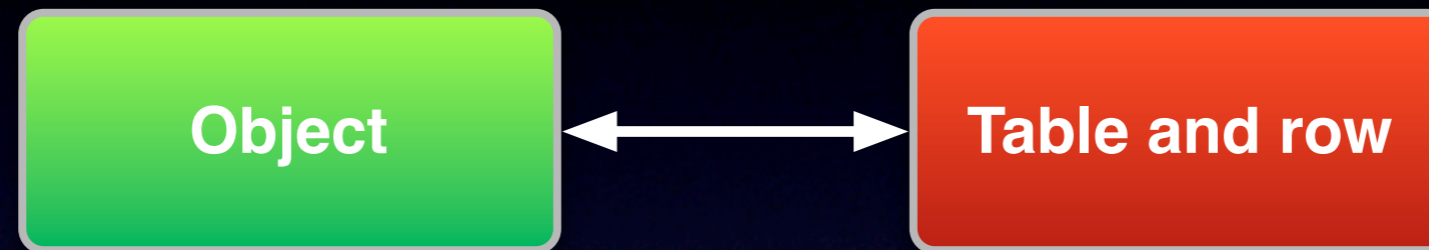
result.close()
conn.close()
```

- The expression language isn't an afterthought for one-off sql statements
- build programmatic SQL using Python

# ORM

- The ORM builds on the Expression Language
- Allows mapping “plain” Python objects - no special inheritance, to database
- SA’s orm is based on the *Data Mapper* design pattern

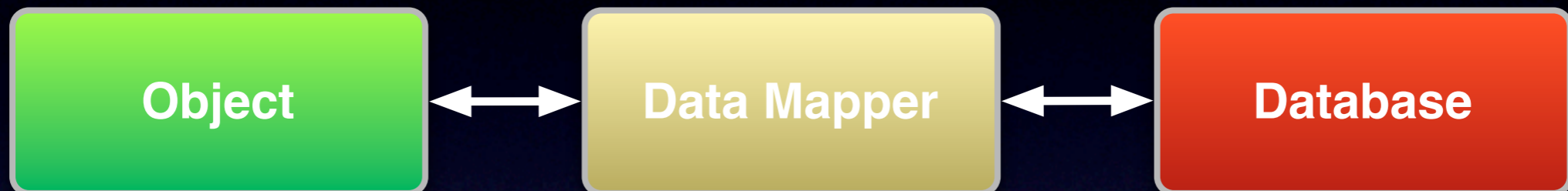
# ORM: Active Record



- Every object is assumed to correspond directly to a table and a row
- The object contains methods for CRUD (Create, Read, Uppdate, Delete)
- Strong mirroring between database schema and object model definition

Active record is the pattern that Rails and Django use for their ORMs.

# ORM: Data Mapper



- How your objects map to the database are controlled by mapper objects
- There is no requirement for a 1:1, table & row to object mapping
- Active Record is *kind of* like a special case of Data Mapper

# Why Data Mapper

- More flexible
- Decouples object and application logic from the database representation
- Allows objects to represent complex data ops (joins, arbitrary selects, sql functions)
- Easier to build apps on legacy databases
- Can operate like Active Record

# Why *not* Data Mapper

- Database schema / model definition synchronization not strictly enforced by the ORM (could be a good or bad thing)
- More 'things' to think about (sessions or object repositories)
- For simple CRUD apps, might be overkill

# session operations

- The center of the SQLAlchemy ORM is the Session
- Provides transactions, and Unit-Of-Work pattern
- Talks to the mappers, handles obj <-> db

# session example

```
session = Session()
session.query(NewsPaper).all()
session.query(Article).all()

article = Article(title="The Great Story")
newspaper = NewsPaper(name="The Globe")

newspaper.articles.append(article)

session.add(newspaper)
session.commit()
```

- Open a session, operate on ORM mapped Python classes
- Add objects to the session then save them by committing the session

# Unit of Work

- Keeps a list of all objects that have been modified and coordinates the writing of all changes to the persistent store
- Uses an identity map, based on primary keys, that allows it to track objects
- Primary use: helps avoid lots of small and/or unnecessary database calls

# Schema Reflection

- Using the option of `autoload=True` on a table, SQLAlchemy will build its schema from the database
- **Very** useful for building apps on existing databases

Demo

Plug!



smarterer

# Smarterer is hiring!

We're looking for deeply passionate technical people. Python, web dev, cloud, full stack.

[mikepk@smarterer.com](mailto:mikepk@smarterer.com)