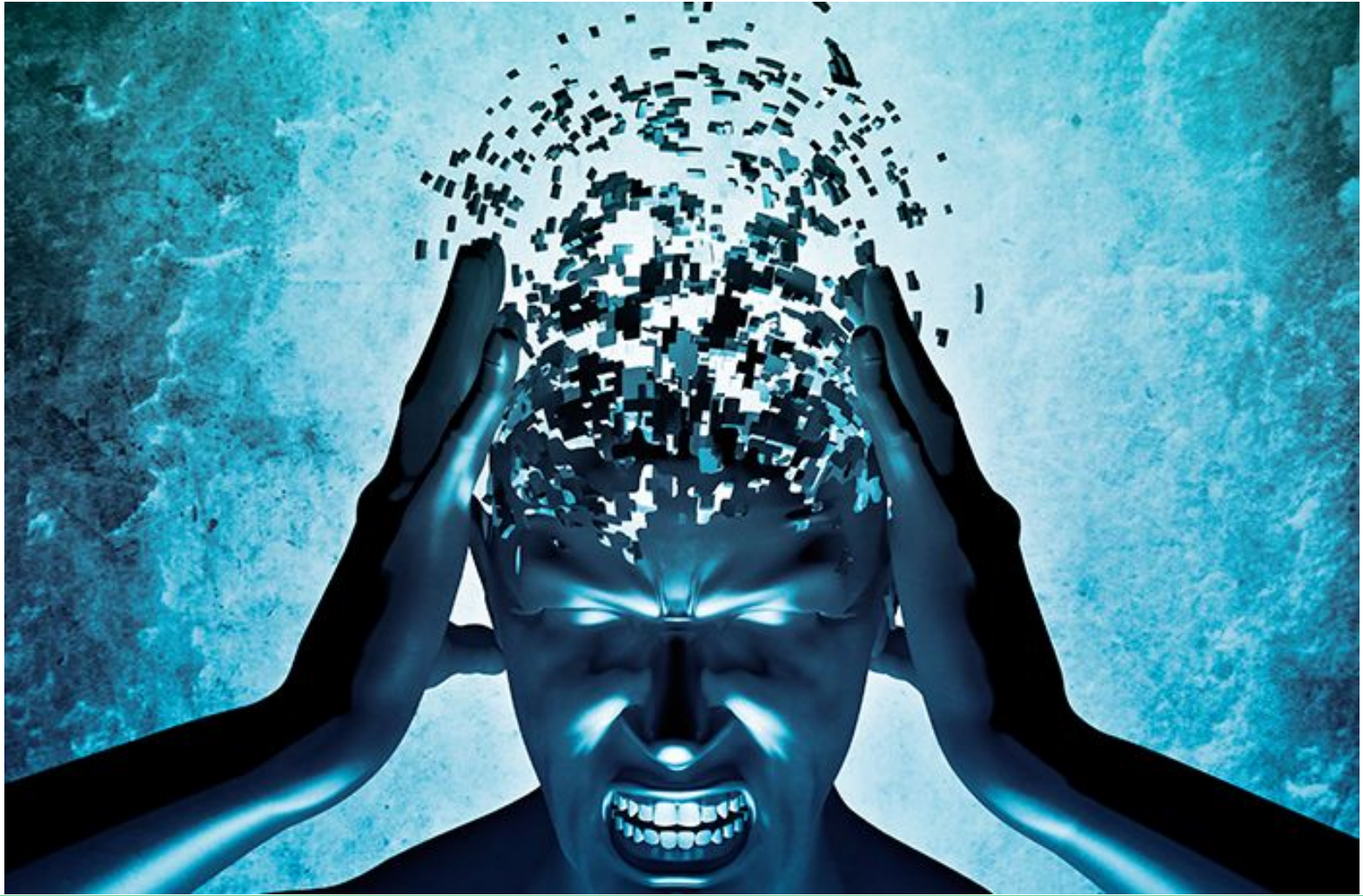
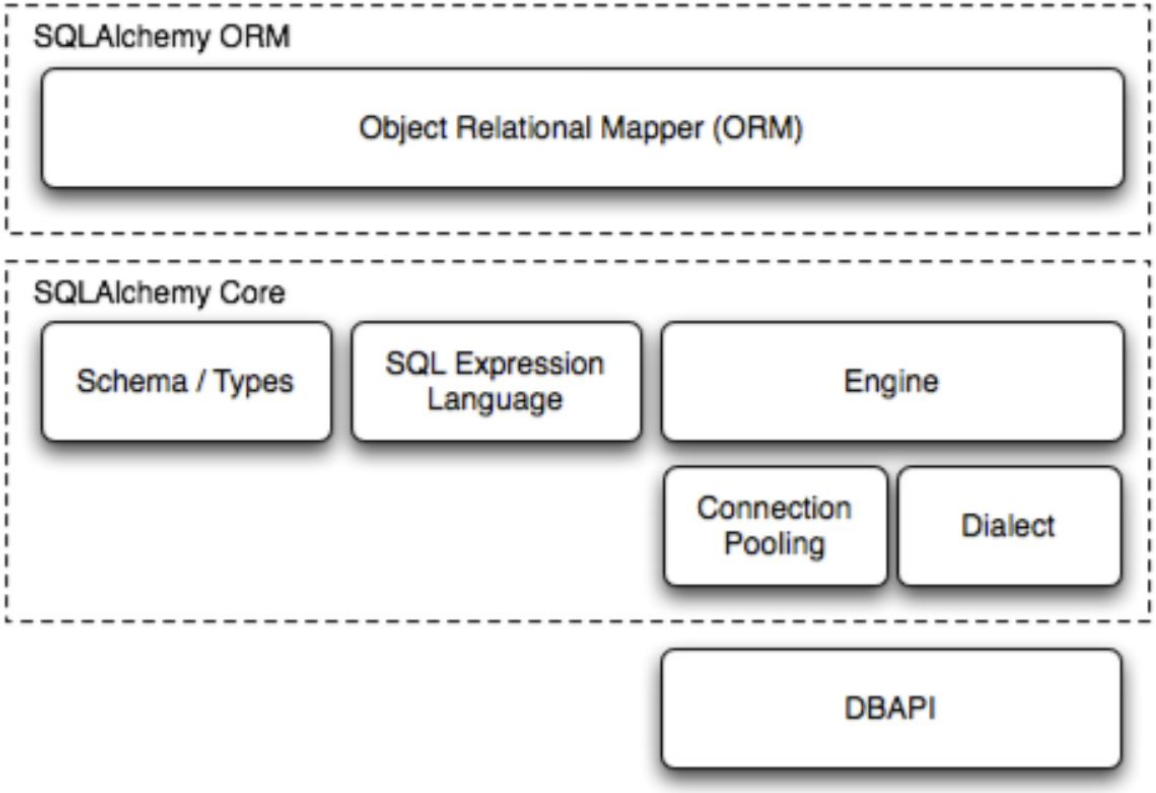
SQLAlchemy

Friend or foe?

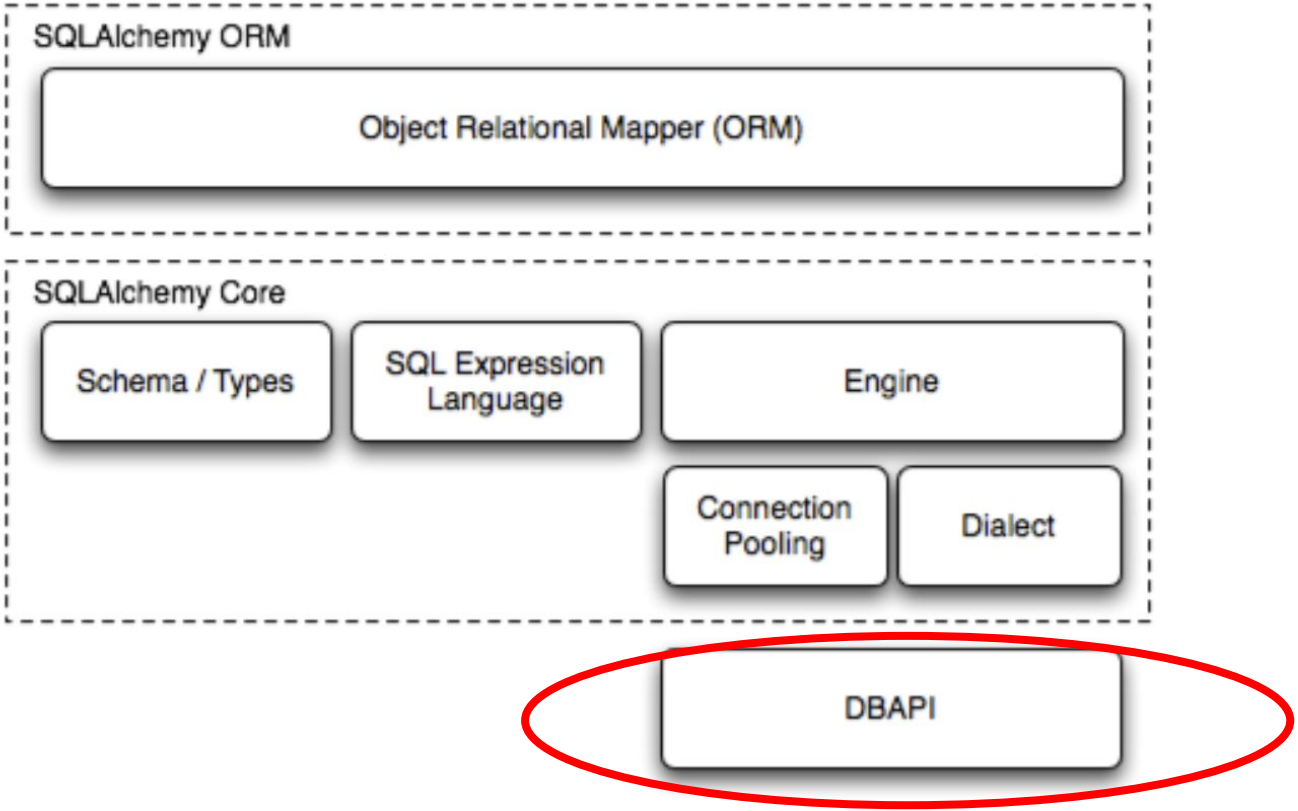


Overall Architecture



DBAPI

DBAPI


SQLAlchemy ORM
Object Relational Mapper (ORM)
SQLAlchemy Core
Schema / Types
SQL Expression Language
Engine
Connection Pooling
Dialect
DBAPI

DBAPI

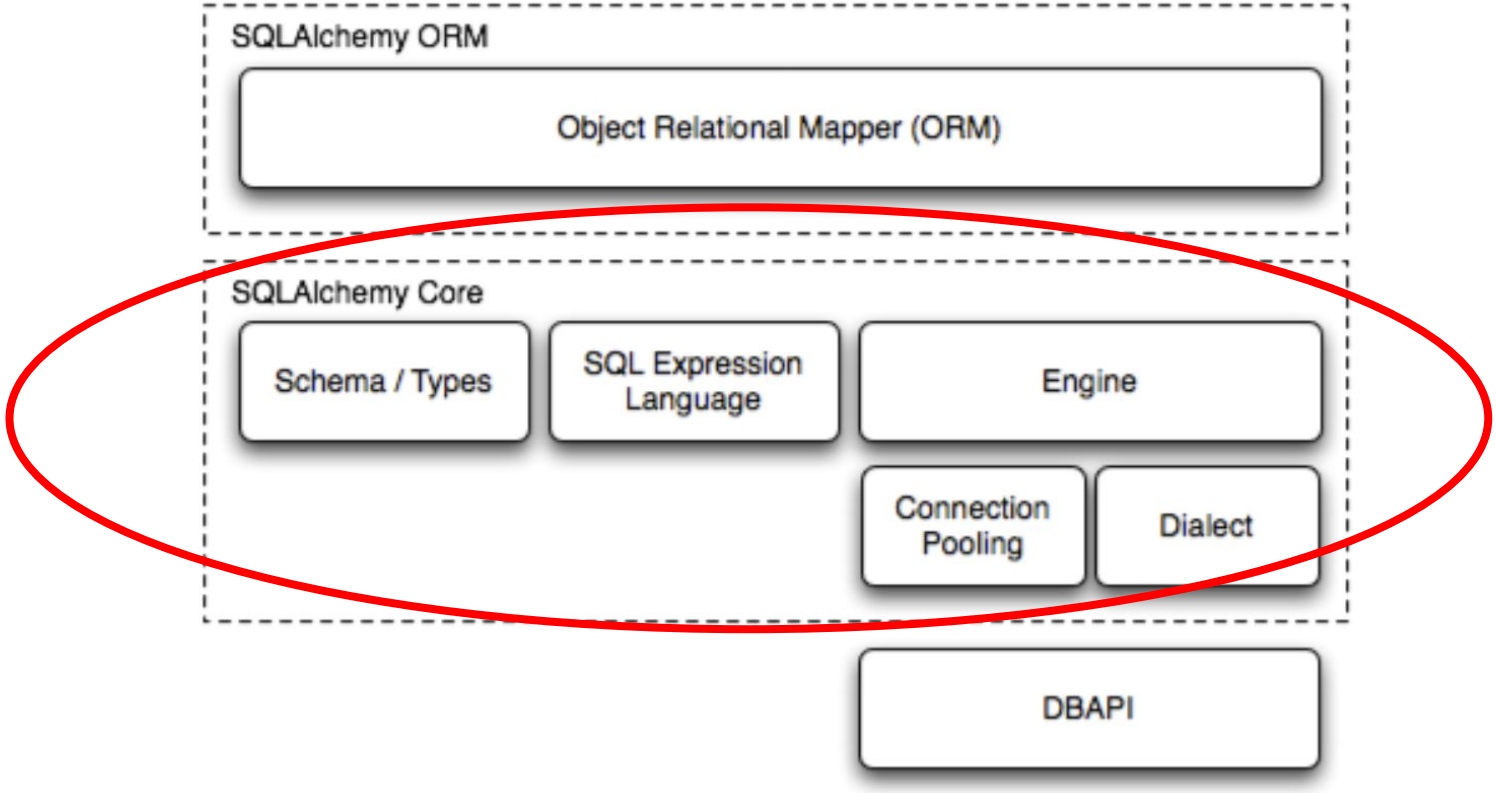
- A standard specification for the API of Python modules providing DB interactions
- Defined by Pep 249
- The lowest-level of abstraction in the SqlAlchemy stack

DBAPI - example

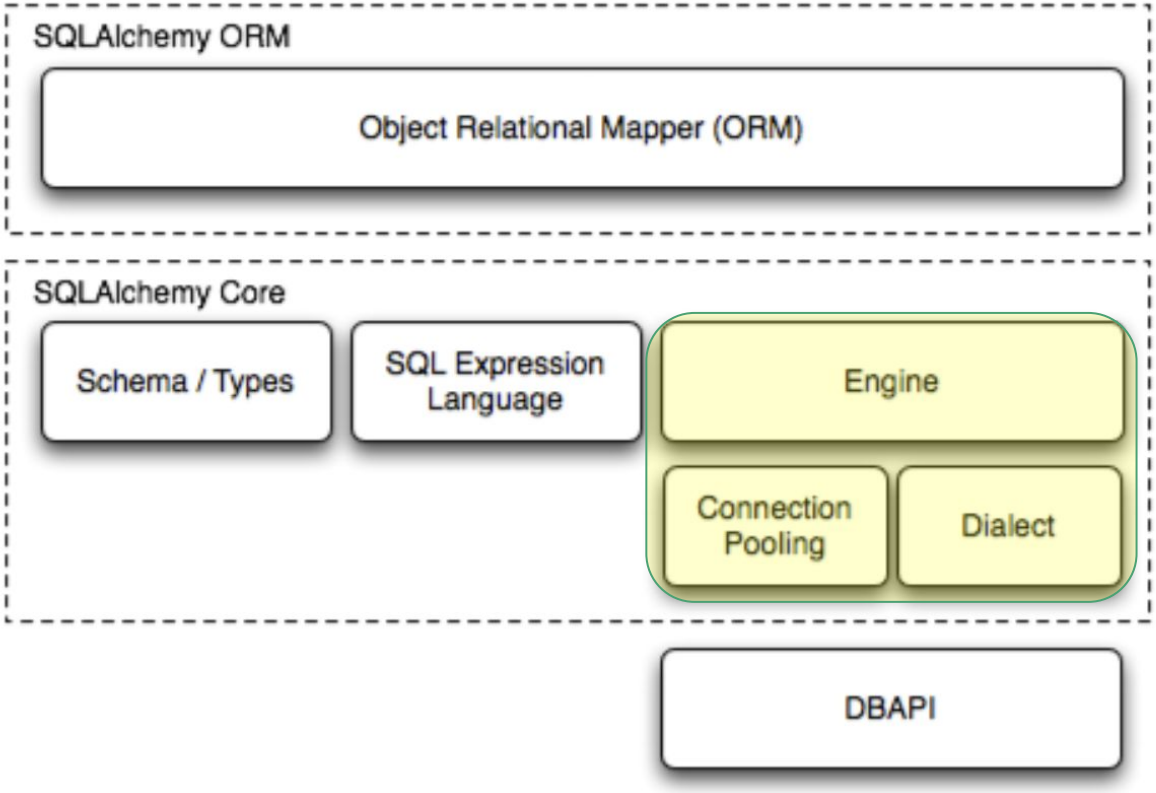
```
In [3]: connection = psycopg2.connect('dbname=db user=jasonkillian port=1612 host=localhost')

In [4]: cursor = connection.cursor()

In [5]: cursor.execute('select * from users')

In [6]: cursor.fetchone()
Out[6]:
(743,
 '743@trialspark.com',
 None,
 None,
 None,
 datetime.datetime(2017, 9, 5, 17, 10, 57, 802335),
 datetime.datetime(2019, 2, 18, 16, 53, 11, 169607),
 False,
 True,
 '_Douglas',
 '_Mays',
 'TRIALSPARK_TEAM_MEMBER',
 datetime.datetime(2019, 7, 30, 17, 27, 53, 823401))
```

SQLAlchemy Core

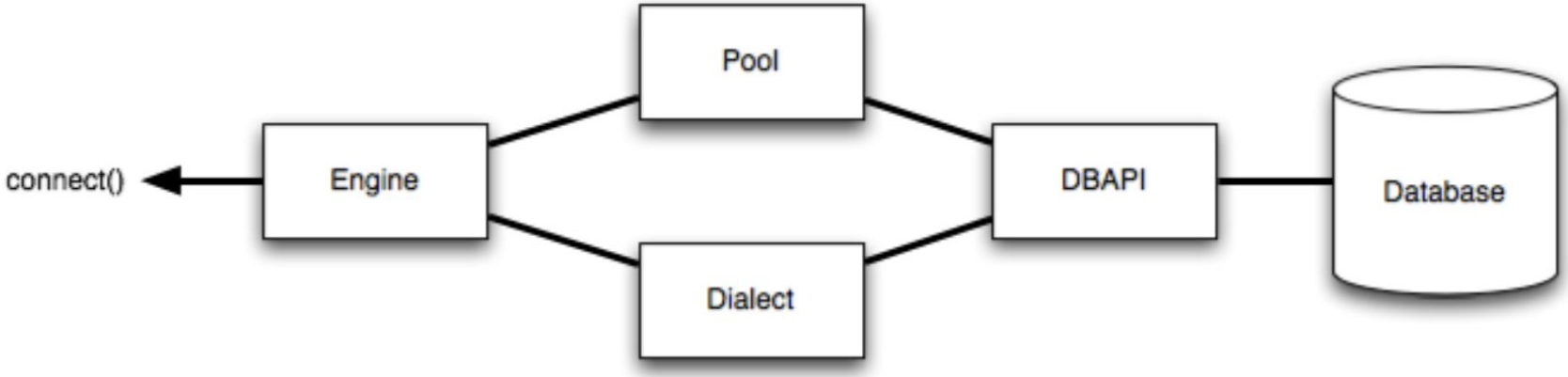
SQLAlchemy Core



SQLAlchemy Core - Engine



SQLAlchemy Core - Engine

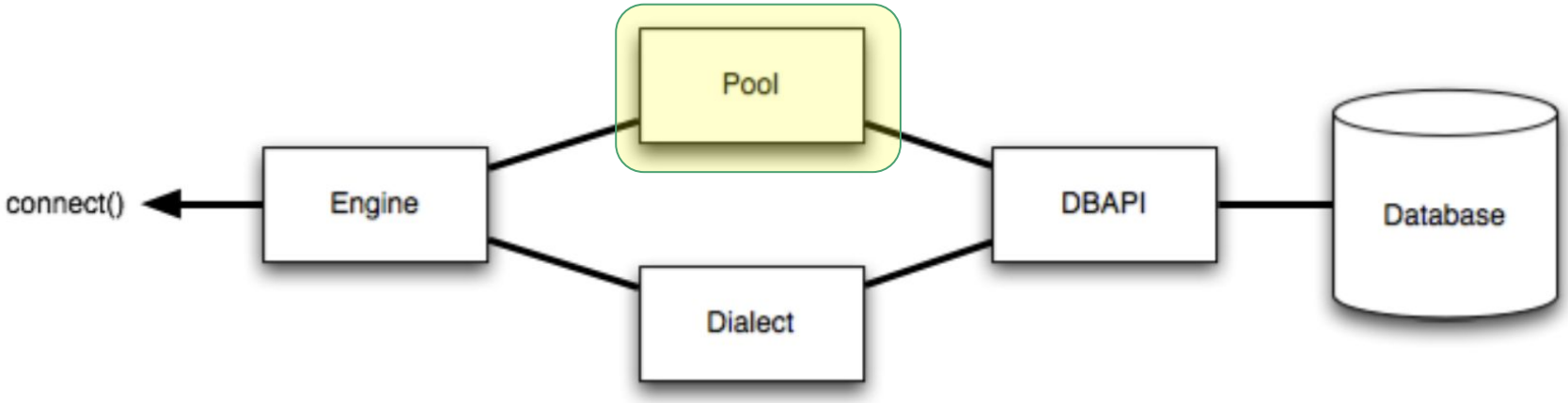


SQLAlchemy Core - Engine

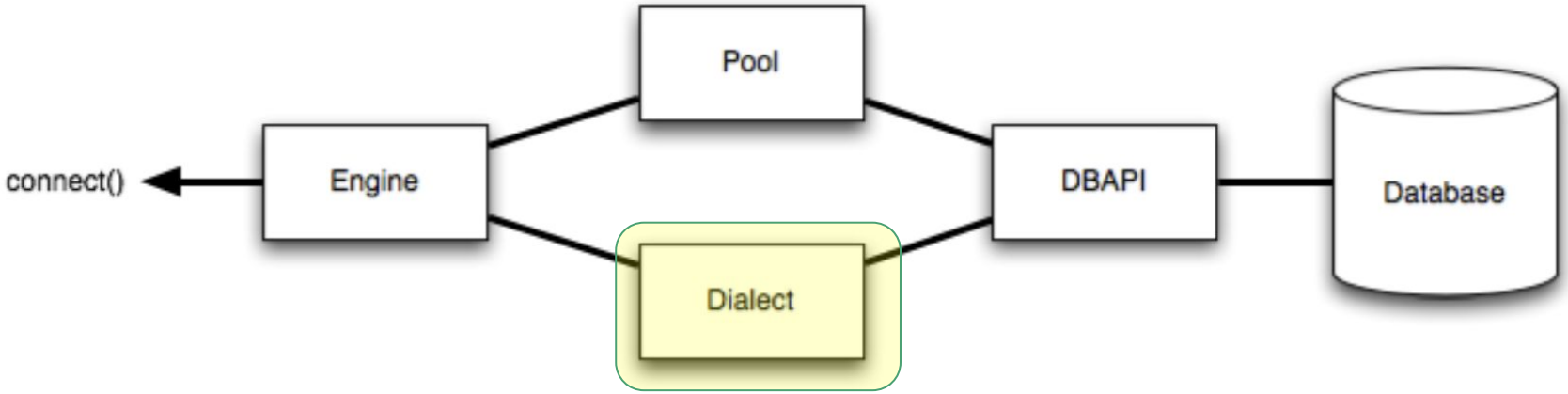
```
engine = create_engine('mysql://scott:tiger@localhost/test')
```

```
connection = engine.connect()
result = connection.execute("select username from users")
for row in result:
    print("username:", row['username'])
connection.close()
```

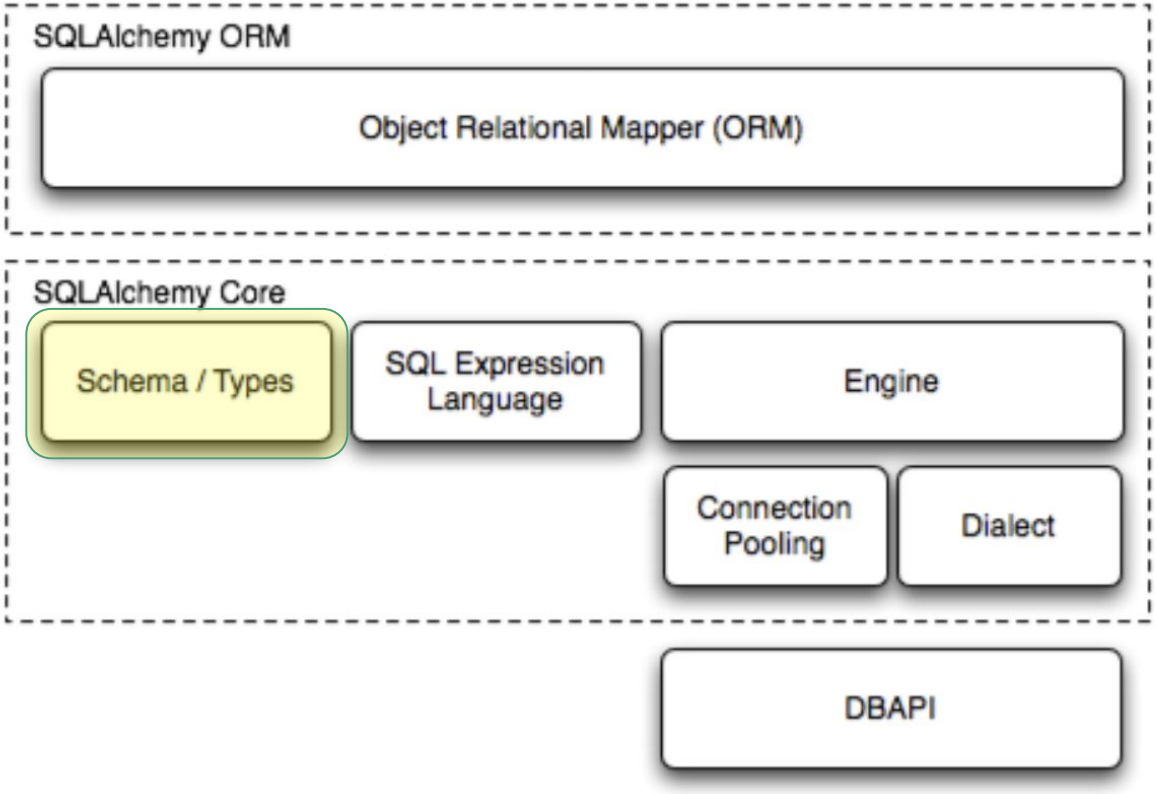
SQLAlchemy Core - Engine - Connection Pooling



SQLAlchemy Core - Engine - Dialect



SQLAlchemy Core - Schema/Types



SQLAlchemy Core - Schema/Types - Declaring

```
from sqlalchemy import *

metadata = MetaData()
```

```
employees = Table('employees', metadata,
    Column('employee_id', Integer, primary_key=True),
    Column('employee_name', String(60), nullable=False),
    Column('employee_dept', Integer, ForeignKey("departments.department_id"))
)
```

SQLAlchemy Core - Schema/Types - Inspecting

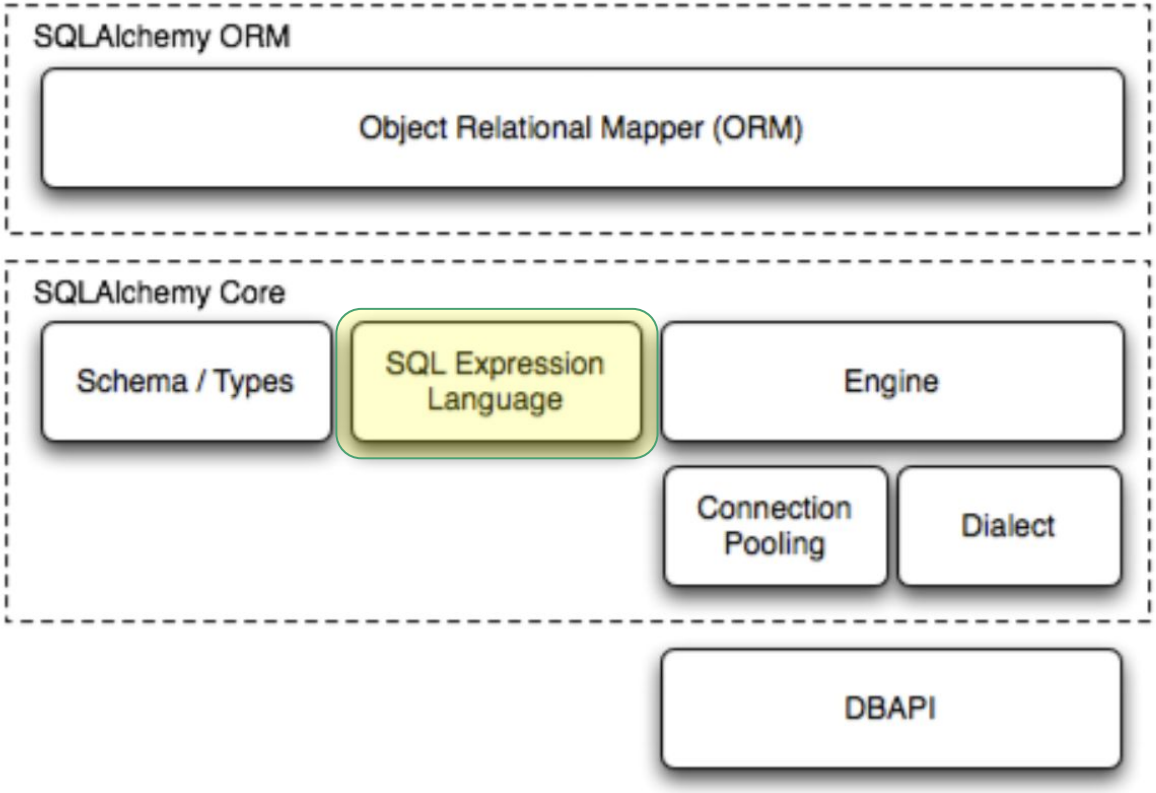
```
# iterate through all columns
for c in employees.c:
    print(c)
```

```
# access a column's name, type, nullable, primary key, foreign key
employees.c.employee_id.name
employees.c.employee_id.type
employees.c.employee_id.nullable
employees.c.employee_id.primary_key
employees.c.employee_dept.foreign_keys
```

SQLAlchemy Core - Schema/Types - Reflecting

```
>>> messages = Table('messages', meta, autoload=True, autoload_with=engine)
>>> [c.name for c in messages.columns]
['message_id', 'message_name', 'date']
```

SQLAlchemy Core - SQL Expression Language



SQLAlchemy Core - SQL Expression Language

```
In [7]: from sqlalchemy import *

In [8]: meta = MetaData()

In [9]: t = Table('users', meta, Column('id', Integer, primary_key=True))

In [10]: t.columns
Out[10]: <sqlalchemy.sql.base.ImmutableColumnCollection at 0x10db7ed38>

In [11]: t.c['id']
Out[11]: Column('id', Integer(), table=<users>, primary_key=True, nullable=False)

In [12]: t.c['id'] == 1
Out[12]: <sqlalchemy.sql.elements.BinaryExpression object at 0x10e5636a0>

In [13]: str(t.c['id'] == 1)
Out[13]: 'users.id = :id_1'
```

SQLAlchemy Core - SQL Expression Language

```
In [16]: t.select().where(t.c['id'] > 10)
Out[16]: <sqlalchemy.sql.selectable.Select at 0x10e562358; Select object>

In [17]: str(t.select().where(t.c['id'] > 10))
Out[17]: 'SELECT users.id \nFROM users \nWHERE users.id > :id_1'
```

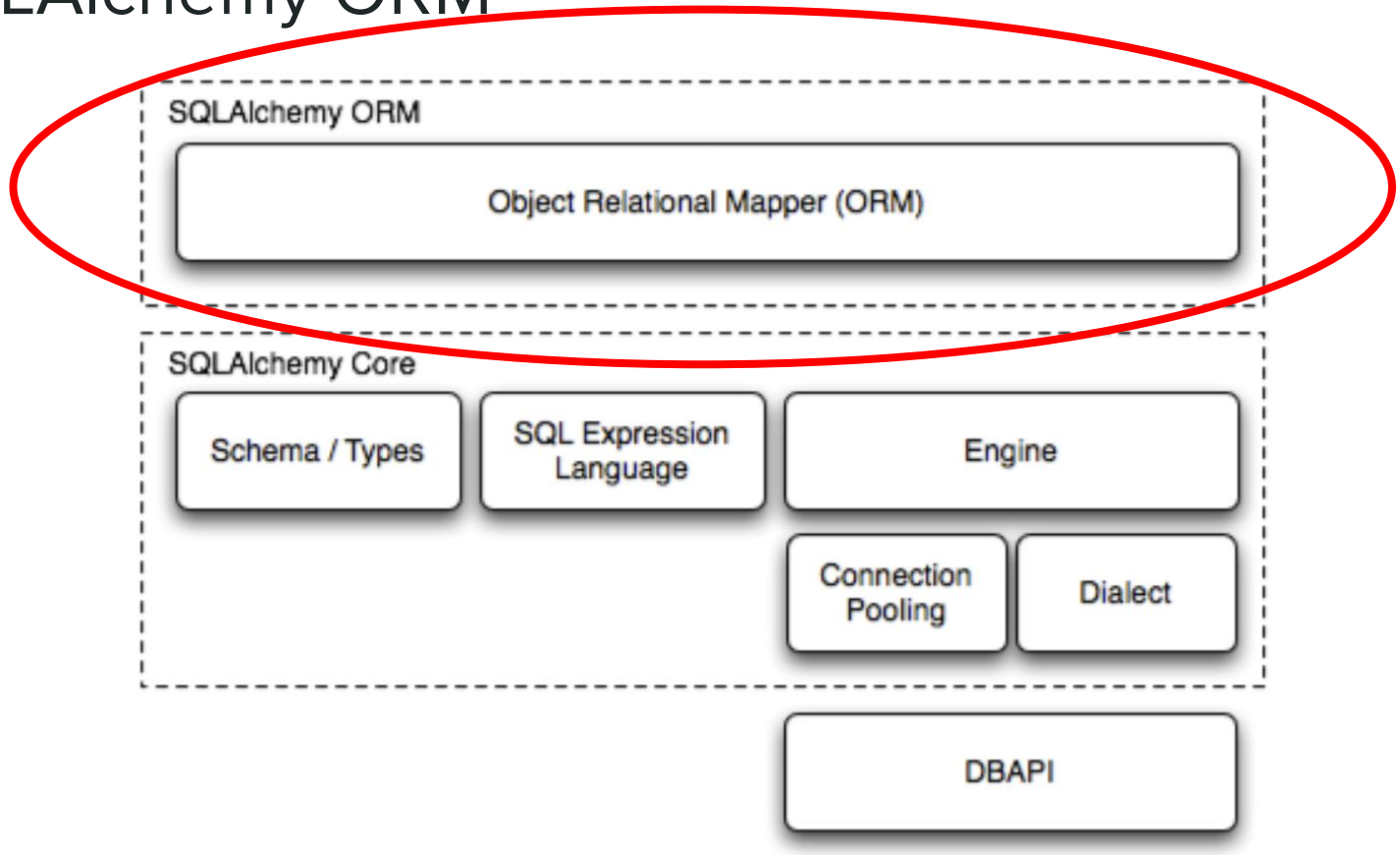
```
In [23]: str(select([t]).where(t.c['id'] > 10))
Out[23]: 'SELECT users.id \nFROM users \nWHERE users.id > :id_1'

In [24]: statement = select([t]).where(t.c['id'] > 10)

In [25]: conn.execute(statement)
```

SQLAlchemy Core - SQL Expression Language

```
>>> print(and_(
...         users.c.name.like('j%'),
...         users.c.id == addresses.c.user_id,
...         or_(
...              addresses.c.email_address == 'wendy@aol.com',
...              addresses.c.email_address == 'jack@yahoo.com'
...         ),
...         not_(users.c.id > 5)
...       )
...   )
users.name LIKE :name_1 AND users.id = addresses.user_id AND
(addresses.email_address = :email_address_1
   OR addresses.email_address = :email_address_2)
AND users.id <= :id_1
```

SQLAlchemy ORM

SQLAlchemy ORM



SQLAlchemy ORM - Data Mapping - Original

```
from sqlalchemy import Table, MetaData, Column, Integer, String, ForeignKey
from sqlalchemy.orm import mapper

metadata = MetaData()

user = Table('user', metadata,
            Column('id', Integer, primary_key=True),
            Column('name', String(50)),
            Column('fullname', String(50)),
            Column('nickname', String(12))
        )

class User(object):
    def __init__(self, name, fullname, nickname):
        self.name = name
        self.fullname = fullname
        self.nickname = nickname

mapper(User, user)
```

SQLAlchemy ORM - Data Mapping - New

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String, ForeignKey

Base = declarative_base()

class User(Base):
    __tablename__ = 'user'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)
    nickname = Column(String)
```

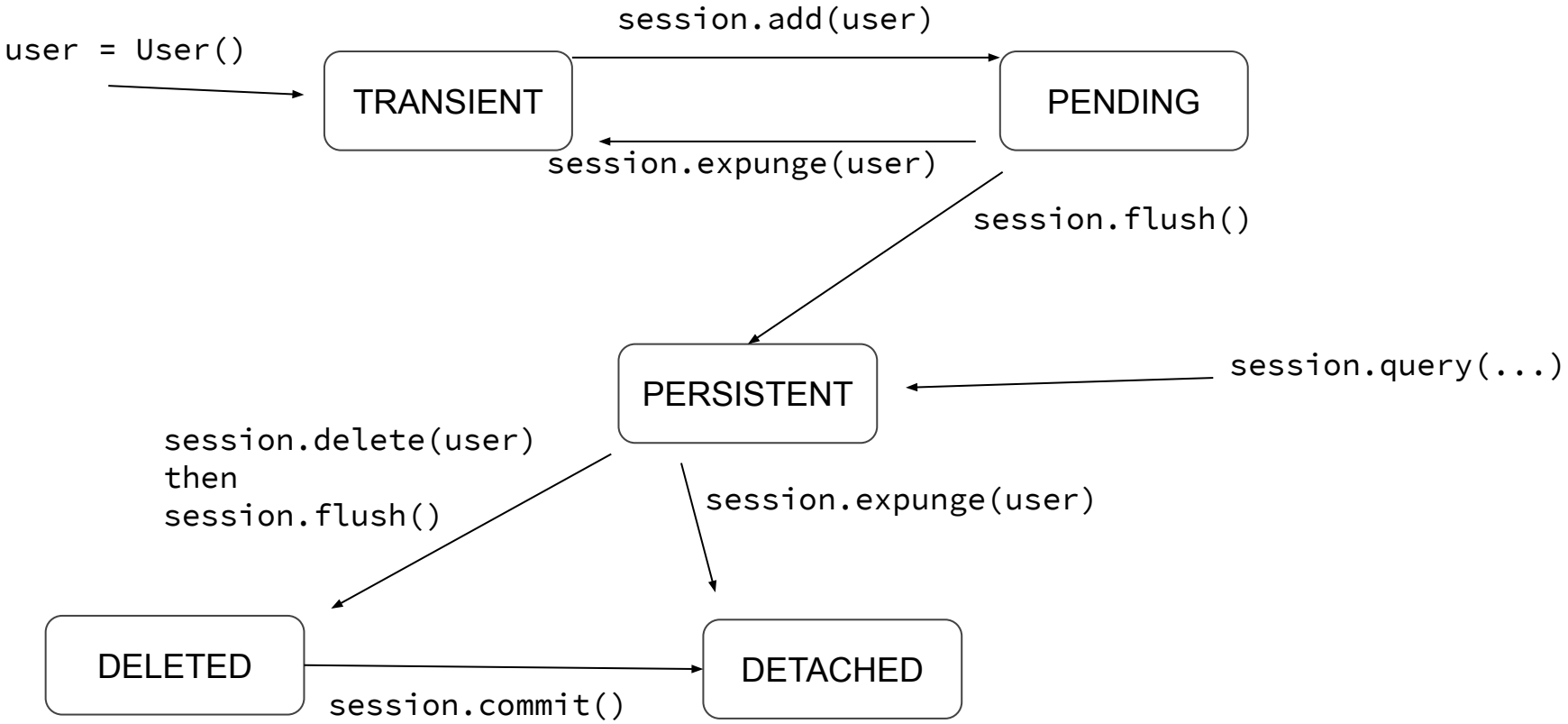
SQLAlchemy ORM - Data Mapping - Session

```
user1 = User(name='user1')
user2 = User(name='user2')
session.add(user1)
session.add(user2)

session.commit()     # write changes to the database
```

SQLAlchemy ORM - Data Mapping - Identity Map

```
In [38]: flag = FeatureFlag(name='TEST_FLAG')

In [39]: session.add(flag)

In [40]: session.identity_map
Out[40]: <sqlalchemy.orm.identity.WeakInstanceDict at 0x119e3add8>

In [41]: session.identity_map.values()
Out[41]: []

In [42]: session.flush()

In [43]: session.identity_map.values()
Out[43]: [<AuditTransaction 1548133>, <FeatureFlag 77>]

In [44]: flag2_maybe = session.query(FeatureFlag).filter_by(id=77).one()

In [45]: flag is flag2_maybe
Out[45]: True
```

SQLAlchemy ORM - Data Mapping - Object States



SQLAlchemy ORM - Data Mapping - Session State

```
# pending objects recently added to the Session
session.new

# persistent objects which currently have changes detected
# (this collection is now created on the fly each time the property is called)
session.dirty

# persistent objects that have been marked as deleted via session.delete(obj)
session.deleted

# dictionary of all persistent objects, keyed on their
# identity key
session.identity_map
```

SQLAlchemy ORM - Data Mapping - expire/refresh

```
In [56]: flag.__dict__
Out[56]:
{'_sa_instance_state': <sqlalchemy.orm.state.InstanceState at 0x119f116a0>,
 'updated_at': datetime.datetime(2019, 10, 3, 4, 25, 51, 714691),
 'globally_enabled': False,
 'id': 77,
 'created_at': datetime.datetime(2019, 10, 3, 4, 25, 51, 714683),
 'name': 'TEST_FLAG'}

In [57]: inspect(flag).persistent
Out[57]: True

In [58]: session.commit()

In [59]: flag.__dict__
Out[59]: {'_sa_instance_state': <sqlalchemy.orm.state.InstanceState at 0x119f116a0>}

In [60]: inspect(flag).persistent
Out[60]: True

In [61]: flag.name
Out[61]: 'TEST_FLAG'
```

Interlude - Flask SQLAlchemy

- Main responsibility is to set up a session for each request and close/teardown the session at the end of the request
- Also adds some convenience utilities to SQLAlchemy like direct `Model.query` syntax via a custom declarative base class

SQLAlchemy ORM Relationships

SQLAlchemy ORM - Relationships

```
class Parent(Base):
    __tablename__ = 'parent'
    id = Column(Integer, primary_key=True)
    children = relationship("Child", back_populates="parent")

class Child(Base):
    __tablename__ = 'child'
    id = Column(Integer, primary_key=True)
    parent_id = Column(Integer, ForeignKey('parent.id'))
    parent = relationship("Parent", back_populates="children")
```

SQLAlchemy ORM - Relationships - Joining vs Loading

```
In [16]: staff = session.query(Staff).join(User).filter(Staff.id == 848).one()
2019-10-03 11:22:47,689 INFO sqlalchemy.engine.base.Engine SELECT pilot.staff.created_at AS pilot_sta
ff_updated_at, pilot.staff.id AS pilot_staff_id, pilot.staff.first_name AS pilot_staff_first_name, pi
t.staff.user_id AS pilot_staff_user_id, pilot.staff.role_type AS pilot_staff_role_type, pilot.staff.p
ntity, pilot.staff.phone AS pilot_staff_phone, pilot.staff.cell_phone AS pilot_staff_cell_phone, pilo
t_email AS pilot_staff_payment_email, pilot.staff.gender AS pilot_staff_gender
FROM pilot.staff JOIN users ON users.id = pilot.staff.user_id
WHERE pilot.staff.id = %(id_1)s
2019-10-03 11:22:47,689 INFO sqlalchemy.engine.base.Engine {'id_1': 848}

In [17]: staff.user
2019-10-03 11:23:11,237 INFO sqlalchemy.engine.base.Engine SELECT users.password AS users_password, u
_at AS users_updated_at, users.id AS users_id, users.email AS users_email, users.last_password_change
AS users_reset_token, users.reset_expires_at AS users_reset_expires_at, users.legacy AS users_legacy,
name AS users_last_name, users.active AS users_active, users.user_type AS users_user_type
FROM users
WHERE users.id = %(param_1)s
2019-10-03 11:23:11,237 INFO sqlalchemy.engine.base.Engine {'param_1': 2917}
Out[17]: <User 2917>
```

SQLAlchemy ORM - Relationships - Joining vs Loading

```
In [20]: staff = session.query(Staff).options(selectinload('user')).filter(Staff.id == 847).one()
2019-10-03 11:25:23,442 INFO sqlalchemy.engine.base.Engine SELECT pilot.staff.created_at AS pilot_s
ff_updated_at, pilot.staff.id AS pilot_staff_id, pilot.staff.first_name AS pilot_staff_first_name,
t.staff.user_id AS pilot_staff_user_id, pilot.staff.role_type AS pilot_staff_role_type, pilot.staff
ntity, pilot.staff.phone AS pilot_staff_phone, pilot.staff.cell_phone AS pilot_staff_cell_phone, pi
t_email AS pilot_staff_payment_email, pilot.staff.gender AS pilot_staff_gender
FROM pilot.staff
WHERE pilot.staff.id = %(id_1)s
2019-10-03 11:25:23,443 INFO sqlalchemy.engine.base.Engine {'id_1': 847}
2019-10-03 11:25:23,447 INFO sqlalchemy.engine.base.Engine SELECT users.id AS users_id, users.passw
ated_at, users.updated_at AS users_updated_at, users.email AS users_email, users.last_password_chan
AS users_reset_token, users.reset_expires_at AS users_reset_expires_at, users.legacy AS users_legac
name AS users_last_name, users.active AS users_active, users.user_type AS users_user_type
FROM users
WHERE users.id IN (%(primary_keys_1)s)
2019-10-03 11:25:23,448 INFO sqlalchemy.engine.base.Engine {'primary_keys_1': 2916}

In [21]: staff.user
Out[21]: <User 2916>
```

SQLAlchemy ORM - Relationships - Joining vs Loading

```
In [25]: staff = session.query(Staff).options(contains_eager(Staff.user)).join(User).filter(User.id == 2915).one()
2019-10-03 11:29:17,187 INFO sqlalchemy.engine.base.Engine SELECT pilot.staff.created_at AS pilot_staff_created_at,
ff_updated_at, pilot.staff.id AS pilot_staff_id, pilot.staff.first_name AS pilot_staff_first_name, pilot.staff.last_
t.staff.user_id AS pilot_staff_user_id, pilot.staff.role_type AS pilot_staff_role_type, pilot.staff.polymorphic_iden
ntity, pilot.staff.phone AS pilot_staff_phone, pilot.staff.cell_phone AS pilot_staff_cell_phone, pilot.staff.pin AS
t_email AS pilot_staff_payment_email, pilot.staff.gender AS pilot_staff_gender, users.password AS users_password, us
 users.updated_at AS users_updated_at, users.id AS users_id, users.email AS users_email, users.last_password_change
s.reset_token AS users_reset_token, users.reset_expires_at AS users_reset_expires_at, users.legacy AS users_legacy,
e, users.last_name AS users_last_name, users.active AS users_active, users.user_type AS users_user_type
FROM pilot.staff JOIN users ON users.id = pilot.staff.user_id
WHERE users.id = %(id_1)s
2019-10-03 11:29:17,187 INFO sqlalchemy.engine.base.Engine {'id_1': 2915}

In [26]: staff.user
Out[26]: <User 2915>
```

SQLAlchemy ORM - Relationships - Updates

```
In [51]: subject = session.query(Subject).get(8)

In [52]: address = subject.address

In [53]: address.subject
Out[53]: <Subject 8>

In [54]: subject.address_id = None

In [55]: address.subject
Out[55]: <Subject 8>

In [56]: session.flush()

In [57]: address.subject
Out[57]: <Subject 8>

In [58]: session.commit()

In [59]: address.subject
```

SQLAlchemy ORM - Relationships - Deleting

```
In [75]: subject = session.query(Subject).get(13)

In [76]: address = subject.address

In [77]: session.delete(address)

In [78]: subject.address
Out[78]: <Address 37>

In [79]: session.flush()

In [80]: subject.address
Out[80]: <Address 37>

In [81]: session.commit()

In [82]: subject.address

In [83]:
```

SQLAlchemy ORM - Relationships - Cascades

SQLAlchemy ORM - Events

SQLAlchemy-Continuum

SQLAlchemy-Continuum

```
self.session_listeners = {
    'before_flush': self.before_flush,
    'after_flush': self.after_flush,
    'after_commit': self.clear,
    'after_rollback': self.clear,
}
self.mapper_listeners = {
    'after_delete': self.track_deletes,
    'after_update': self.track_updates,
    'after_insert': self.track_inserts,
}
self.class_config_listeners = {
    'instrument_class': self.builder.instrument_versioned_classes,
    'after_configured': self.builder.configure_versioned_classes,
}
```

Resources

- SQLAlchemy Docs - can be a bit hard to understand, but obviously a great resource for learning the ins and outs of SQLAlchemy
- Introduction to SQLAlchemy - PyCon 2014 - A good but long talk on SQLAlchemy fundamentals from its creator
- The Architecture of Open Source Applications: SQLAlchemy - A nice writeup on the architecture of the SQLAlchemy codebase