The Philosopher's Groan

How I Learned to Love SQLAlchemy

SQL (mostly) *describes* data(bases).

SQL (mostly) *describes* data(bases).

Python expresses anything you want.

SQL (mostly) *describes* data(bases).
Python expresses anything you want.
Python very very frequently expresses data transformation.

SQL (mostly) *describes* data(bases).
Python expresses anything you want.
Python very very frequently expresses data transformation.
Data often comes from (SQL) databases.

SQL (mostly) *describes* data(bases).
Python expresses anything you want.
Python very very frequently expresses data transformation.
Data often comes from (SQL) databases.

Conclusion?

Python needs to talk to SQL Databases.

Python needs to talk to SQL Databases.
Like, all the time. 😳

Object

Relational

Mapping


and black with fate

SQLAlchemy Offers Us 2.5 Ways of Doing Things

SQLAlchemy Offers Us 2.5 Ways of Doing Things

1. sqlalchemy.orm
2. sqlalchemy.core
3. “headless” sqlalchemy.core*

*i made that term up


FUNiMATION

Quick Run Through of Some ORM Code

```python
# ddl
class Baker(Base):
    __tablename__ = "baker"

    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    pronouns = Column(String)


    def __repr__(self):
        return f"<Baker {self.name} ({self.pronouns})>"
```

```
# insert
to_insert = [
    ["Ed", "he/him"], ["Zeb", "they/them"],
    ["Rheta", "she/her"], ["Brad", None],
]
session.add_all(
    [
        Baker(name=name, pronouns=pronouns)
        for name, pronouns in to_insert
    ]
)
session.commit()
```

```
# select and delete
no_pronouns = session.query(Baker).filter(
    Baker.pronouns is None
)
no_pronouns.delete()

session.commit()
```

So why ORM?

So why ORM?

1. You like OOP, ORMs, Python, or any combo of the 3.
2. Your i/o is straightforward, your db is clean.
3. You like Django.
4. You want predictable code.

Quick Run Through of Common Core Code

```
# ddl
baker = Table(
    "baker",
    metadata,
    Column("id", Integer, primary_key=True),
    Column("name", String, nullable=False),
    Column("pronouns", String),
)
```

```
# insert
to_insert = [
    ["Ed", "he/him"], ["Zeb", "they/them"],
    ["Rheta", "she/her"], ["Brad", None],
]
insert_statement = baker.insert().values(
    [
        {"name": name, "pronouns": pronouns}
        for name, pronouns, dt in to_insert
    ]
)
conn.execute(insert_statement)
```

```
# select and delete
delete_statement = (
    select([baker])
    .where(baker.c.pronouns is None)
    .delete()
)

conn.execute(delete_statement)
```

So why Core?

So why Core?

1. You like FP and native data structures.
2. You dislike Django.
3. Your queries are complicated.
4. You are a native SQL speaker. You speak SQL eloquently. You love SQL.


PeacEclectic

The Engine

```
# common to core and orm
from sqlalchemy import create_engine

engine = create_engine(“db://nice.url)
engine.dispose()
```

The Database Schema

```python
# core
from sqlalchemy import MetaData
metadata = MetaData(bind=engine)

#orm
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base(bind=engine)
Base.metadata
```

Table Abstractions

```
# core
Table()

# orm
declarative_base()

mapper()

# common
Column()


# headless
metadata.tables[“table_name”]
```

```python
class Baker(Base):
    __tablename__ = "baker"

    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    pronouns = Column(String)
    specialties = relationship(
        "Bread",
        secondary=baker_specialty,
        back_populates="specializing_bakers"
    )

    def __repr__(self):
        return f"<Baker {self.name} {self.pronouns}>"
```

```
baker = Table(
    "baker",
    metadata,
    Column("id", Integer, primary_key=True),
    Column("name", String, nullable=False),
    Column("pronouns", String),
)
```

```
metadata = MetaData(bind=engine)
metadata.reflect_all()

bakery = metadata.tables["baker"]
```

Connections & Transactions

```python
# orm
from sqlalchemy.orm.session import sessionmaker
Session = sessionmaker()  # global
session = Session(bind=engine)  # local
```

```
#notable session attrs
add()
add_all()
delete()
new
dirty
query()
rollback()
commit()
```

```
# core
conn = engine.connect()  # engine: global, conn: local

conn.execute(statement)
conn.close()
```

```
# lesser known conn methods
transaction = conn.begin()
conn.execute(statement)  # not executed till commit
transaction.commit()

raw_conn = engine.raw_connection()  # for barbarians
```

Queries & Queryish Constructs

```python
# creation
wonderbread = Bread(
    name="wonder",
    is_delicious=False,
    ingredient_cost=0.60
)

session.add(wonderbread)

wonderbread.name = "wonderbread"  # session remembers this mutation
session.commit()  # insert emitted now
```

```
# creation
inset_smt = bread.insert().values(
    {"name": "wonder","is_delicious": False, "ingredient_cost": 0.60},
)
conn.execute(insert_smt)

update_smt = (
  bread.update()
   .where(bread.c.name == "wonder")
   .values(name="wonderbread")
)
conn.execute(update_smt)
```

```
talented_lady_bakers = (
    session.query(Baker.name, Bread.name)
    .select_from(Baker)
    .join(baker_specialty)
    .join(Bread)
    .filter(
        Bread.is_delicious == True,
        Baker.pronouns == "she/her",
    )
   .all() # nothing executed on db until here
)
```

```
join = baker.join(baker_specialty).join(bread)
select_smt = (
    select([baker.c.name, bread.c.name])
    .select_from(join)
    .where(
        and_(
            bread.c.is_delicious == True,
            baker.c.pronouns == "she/her",
        )
    )
)
talented_lady_bakers = conn.execute(select_smt).fetchall()
```

```sql
SELECT baker.name, bread.name
FROM baker
JOIN baker_specialty
ON baker.id = baker_specialty.baker_id
JOIN bread
ON bread.id = baker_specialty.bread_id
WHERE bread.is_delicious = true
AND baker.pronouns = 'she/her'
```

What do we make of this?

ORM is an ORM.

There are great conveniences for dealing with:

- i/o spanning database relationships
- business logic
- consistent code

Core is *almost* SQL.

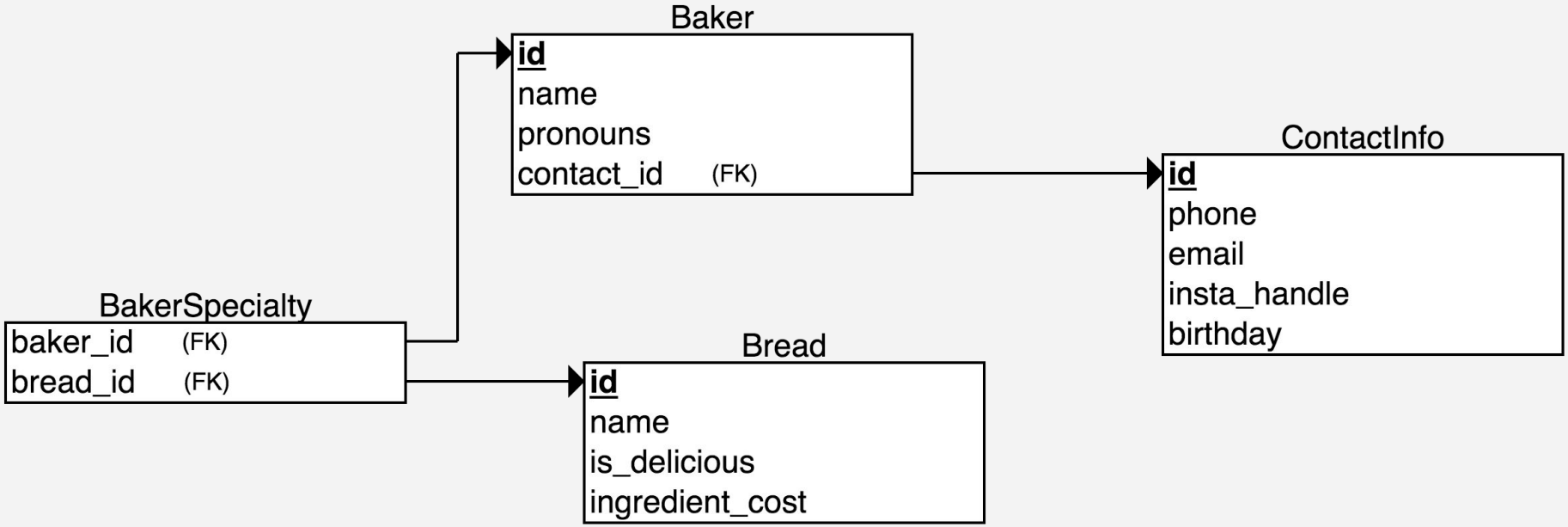
SQL-heads will prefer it: No need to learn new ideas if you're already SQL fluent.

Low-level tools allow for more creative, customized problem solving.

Both may have a place in your codebase.

All the different ways of doing the same thing make it obvious when different kinds of alchemy are happening.


Thanks for listening!
Find me on Twitter @whole_grainsley
& GitHub @ainsleymcgrath


Baker
id
name
pronouns
contact_id (FK)
ContactInfo
id
phone
email
insta_handle
birthday
BakerSpecialty
baker_id (FK)
bread_id (FK)
Bread
id
name
is_delicious
ingredient_cost