# Working With Relational Databases: SQLAlchemy

OOSD

School of Information Technology
Otago Polytechnic
Dunedin, New Zealand

# A very common problem

- We want to work with persistent data.
- A relational database is a very common place to keep such data.
- Moving data between an object oriented model and a relational database is tricky.
- Also, we want to avoid DBMS-specific code in our applications.

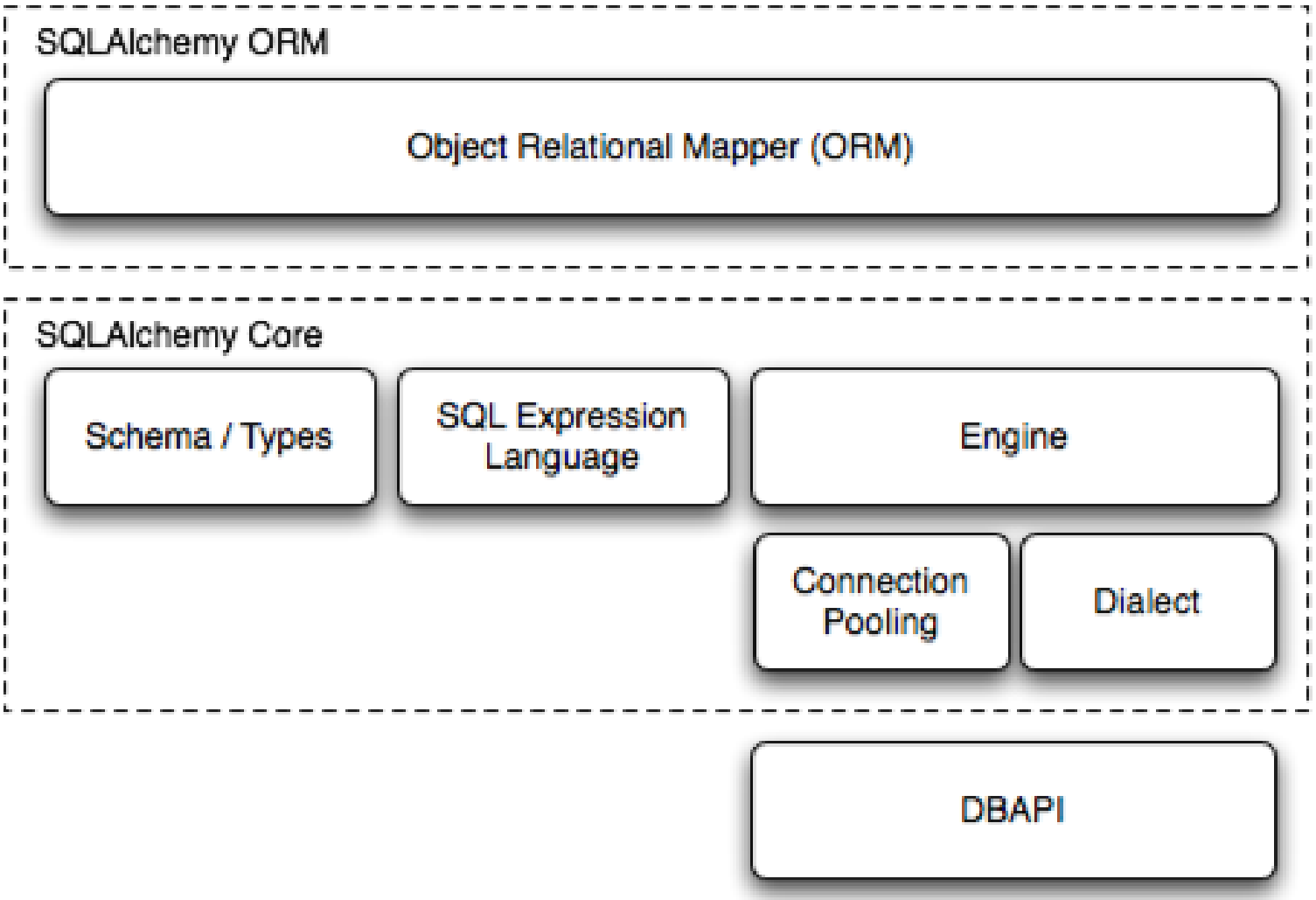
# A very common solution

- There are many libraries for working with databases in every language.
- Some are very nice, some are less so. Most of them work ok.
- A common pattern they implement it *Object-Relational Mapping* (ORM).
- There is, however, a degree of pushback against ORM.

# SQLAlchemy

- SQLAlchemy is an excellent set of libraries for working with databases in Python.
- It supports ORM, but does not require it.
- It's very flexible. Many DB libraries sacrifice flexibility in return for simplicity.

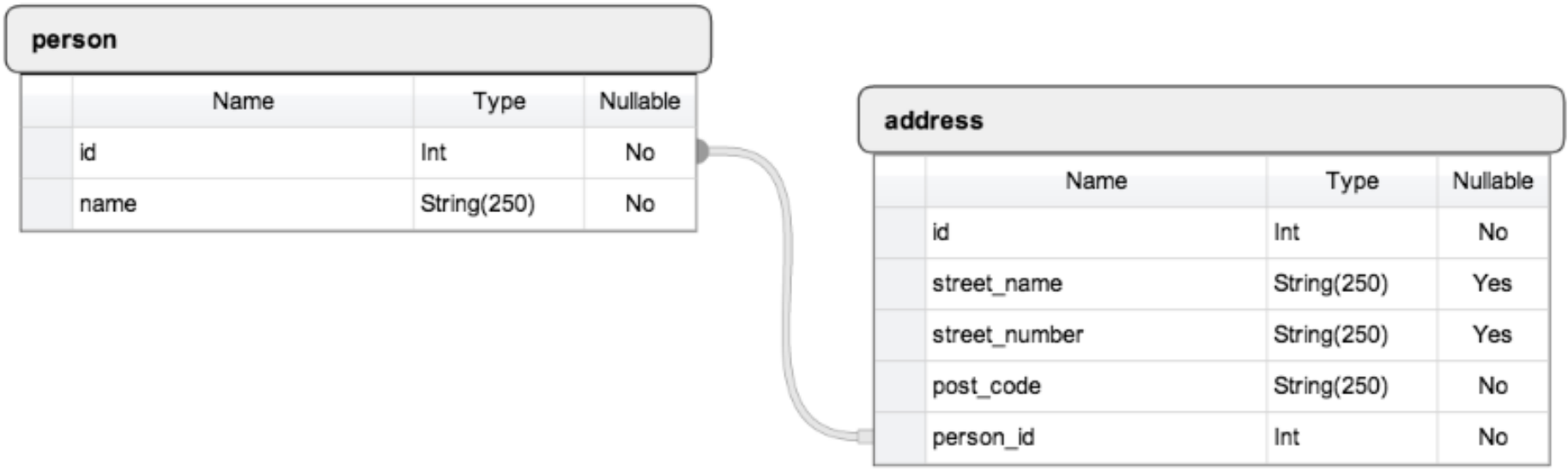
# SQLAlchemy Components



# ORM

- The ORM in SQLAlchemy moves data back-and-forth between application code and a database.
- It converts types between those supported by the DBMS and Python types.

# SQLAlchemy Example



**person**

| | Name | Type | Nullable |
|---|---|---|---|
| | id | Int | No |
| | name | String(250) | No |

**address**

| | Name | Type | Nullable |
|---|---|---|---|
| | id | Int | No |
| | street_name | String(250) | Yes |
| | street_number | String(250) | Yes |
| | post_code | String(250) | No |
| | person_id | Int | No |

# Person class

```
Base = declarative_base()

    class Person(Base):
        __tablename__ = 'person'
        id = Column(Integer, primary_key=True)
        name = Column(String(250), nullable=False)
```

# Address class

```
class Address(Base):
    __tablename__ = 'address'
    id = Column(Integer, primary_key=True)
    street_name = Column(String(250))
    street_number = Column(String(250))
    post_code = Column(String(250), nullable=False)
    person_id = Column(Integer, ForeignKey('person.id'))
    person = relationship(Person)
```

# Creating the database

```
engine = create_engine('sqlite:///sqlalchemy_example.db')
Base.metadata.create_all(engine)
```

# Getting a Record

```
# ... some boilerplate above
session.query(Person).all()
session.query(Person).get(id)
session.query(Person).filter(Person.name == name)
```

# Updating a Record

```
session = DBSession()
# get a Person
# modify a Person
session.add(person)
session.commit()
```

# References

- SQLAlchemy Documentation: `http://docs.sqlalchemy.org/en/rel_1_0/index.html`
- Handy tutorial: `http://www.pythoncentral.io/introductory-tutorial-python-sqlalchemy/`
- Installing SQLALchemy: `http://www.pythoncentral.io/how-to-install-sqlalchemy/`

# Exercise

- Write a set of classes using SQLAlchemy that model students, semesters, and papers. Use SQLite for persistence.
- Write a script that lets me
  - List students
  - List papers
  - Select a paper and see some details for the paper plus a list of students.
  - Select a student and a semester and print the students schedule for the semester.