# Working with Databases and ORM

## Intermediate Application Development

Otago Polytechnic
Dunedin, New Zealand
Kaiako: Tom Clark

# Introduction

Many of the applications that we build and maintain need to store and retrieve data, often in large amounts. Databases, especially relational databases, are one of our primary tools for handling such data. So it's not surprising that we have developed a lot of tooling to help work with relational databases in our code.

The match between objects that hold data and relational database records is not perfect. The tools that help manage the connections between the two are generally described as *Object Relational Mapping* (ORM).

# Functions of DB/ORM Libraries

Although I tend to describe them as ORM libraries, they generally provide a lot of functionality beyond purely ORM uses.

1. Manage connections to a database service
2. Create and modify database tables
3. Handle CRUD operations on objects/tables
4. Abstract out database-specific elements from code

# SQLAlchemy

The most widely used database/ORM library for Python is *SQLAlchemy*. It's divided into core and ORM modules, making it clear that you can use it without any of the ORM features (although we will).

We will use version 1.4 of SQLAlchemy, which includes new features in advance of the 2.0 release. It is not in the Python standard library, so we'll need to use `pip` to install it.

# Connecting: the Engine

```
from sqlalchemy import create_engine

# use an SQLite database file names 'practical21.db'
engine = create_engine('sqlite:///practical21.db',
    future=True)

# use a remote Postgres dbms
engine = create_engine(
    'postgres://user:password@db.op.ac.nz/dbname')
```

The engine does not immediately connect to the database, it just provides the ability to connect when it is needed.

# A Database-Mapped Class

```
from sqlalchemy.orm import declarative_base
from sqlalchemy import Table, Column, Integer, String

Base = declarative_base()

class Cat(Base):
    __tablename__ = 'cats'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    breed = Column(String)
    colour = Column(String)
    age = Column(Integer)

# we can create the table in the database
Base.metadata.create_all(engine)
```

# Creating and Saving Objects

```
from sqlalchemy.orm import Session

lola = Cat(name='Lola', breed='Burmese',
    colour='Black', age=10)
shadow = Cat(name='Shadow', colour='Grey')
leo = Cat(name='Leo', breed='Siamese', age=4)

# to work with the ORM features we need a Session
session = Session(engine)
session.add(lola)
session.add(shadow)
session.add(leo)

# a commit() is necessary to finally save the cats
session.commit()
```

# Querying

```
from sqlalchemy import select

query = select(Cat).where(Cat.name == 'Lola')
# just give us the first matching cat
cat = session.execute(query).first()

query = select(Cat).where(
    Cat.age < 5,
    Cat.colour == 'black')
cats = session.execute(query)
for cat in cats:
    print(cat.name)
```

# Updating

```
query = select(Cat).where(Cat.name == 'Lola')
cat = session.execute(query).first()
cat.age = 11
session.commit()
```

# Deleting

```
query = select(Cat).where(Cat.name == 'Lola')
cat = session.execute(query).first()
session.delete(cat)
session.commit()

# or
from sqlalchemy import delete
session.execute(delete(Cat).where(Cat.name == 'Lola'))
session.execute
```

# Programming Activity

1. Pull the course materials repo.
2. Create a new branch, `21-practical` in your practicals repo.
3. Copy the subdirectory, `21-practical` from the class materials into your repo.
4. See the README for directions.
5. We will discuss results in 20ish minutes.

# More Complex Objects

Suppose we want to record veterinary clinics and associate them with cats.

```
class VetClinic(Base):
  __tablename__ = 'clinics'
  id = Column(Integer, primary_key=True)
  name = Column(String)
  phone_number = Column(String)
```

How do we connect the objects? We want the vet clinic to be an attribute of the Cat class.

# Relationships with Foreign Keys

```
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship

class Cat(Base):
    __tablename__ = 'cats'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    breed = Column(String)
    colour = Column(String)
    age = Column(Integer)
    clinic_id = Column(ForeignKey('clinics.id'))

    clinic = relationship("VetClinic", back_populates='cats')
```

# Using Relationships

Now for a `Cat` object, we can access

```
lola.clinic.phone_number
```

And a `VetClinic` has a list of cats.

```
clinic.cats.append(lola)
```

# References

- SQLAlchemy: (https://www.sqlalchemy.org/)
- SQLAlchemy tutorial (pretty involved): (https://docs.sqlalchemy.org/en/14/tutorial/index.html)