Working with Databases in Python 3

Using an ORM - SQLAlchemy

Douglas Starnes
Author / Speaker

@poweredbyaltnet | linktr.ee/douglasstarnes

Overview

ORM

- Object Relational Mapper
- Creates a mapping between objects and tables

SQLAlchemy

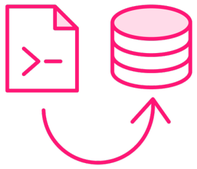
- Core API
- ORM

Model Classes

Relationships

Use the same code with more than one database

What is an ORM?



Object Relational Mapper

Work with a relational database using a general purpose language

Write your entire application in a single language

Connect to multiple databases

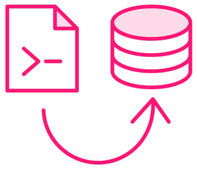
In Previous Episodes

Connect to the database

Send SQL to the server

Wait for the results

What is an ORM?



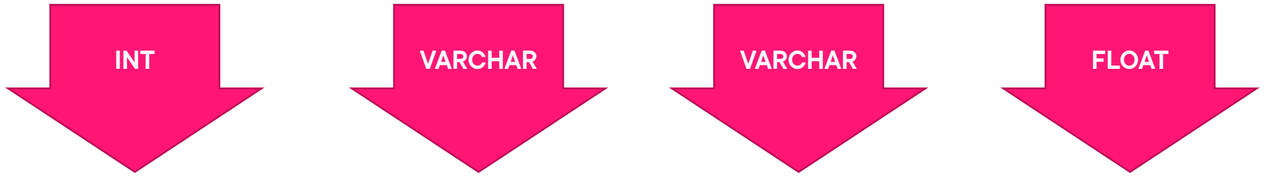
Object Relational Mapper

Work with a relational database using a general purpose language

Write your entire application in a single language

Connect to multiple databases

Object Relational Mapper



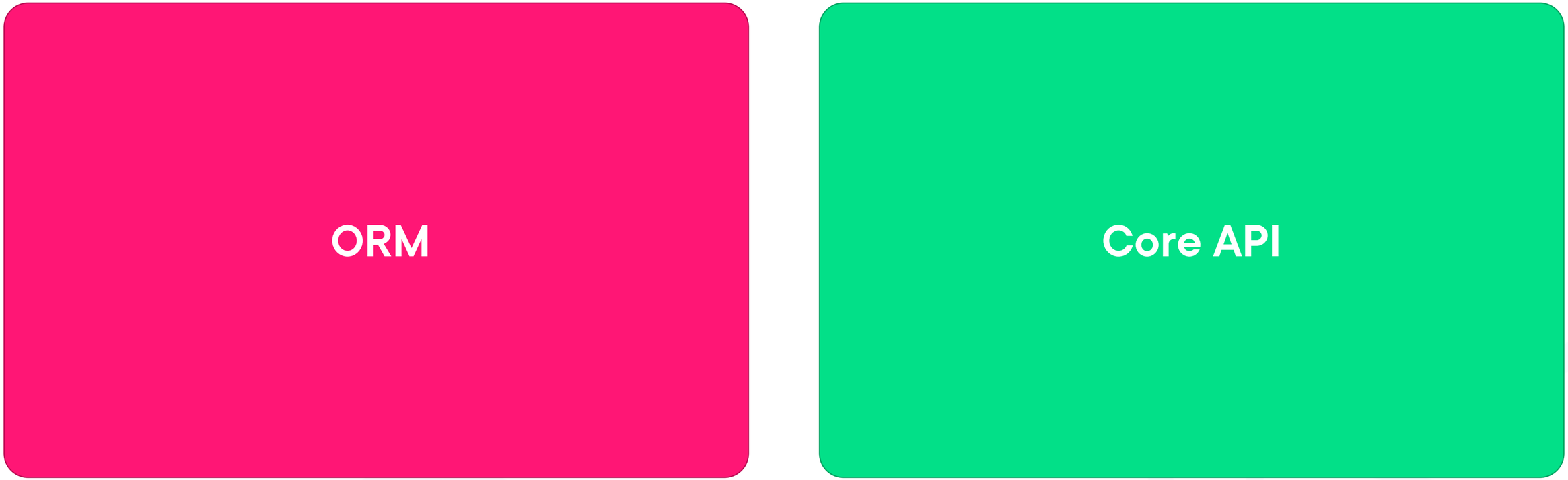
id	coin	currency	amount
1	bitcoin	USD	1.0
2	ethereum	GBP	10.0
3	dogecoin	EUR	100.0

```
INSERT INTO
investment (coin, currency, amount)
VALUES (‘bitcoin’, ‘USD’, 1.0);
```

```
class Investment:
    id: int
    coin: str
    currency: str
    amount: float
```

```
bitcoin = Investment(
    coin=“bitcoin”
    currency=“USD”,
    amount=1.0)
```

Two Sides of the Same API

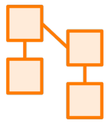


Don’t forget the docs!

https://docs.sqlalchemy.org/en/20/

Using SQLAlchemy

Distributed as a Python package



All model classes must inherit from `DeclarativeBase`

Attribute to column mappings are defining using type annotations

An engine connects to the database

A session is used to interact with the database

Implementing a Model Class

```
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass
```

Implementing a Model Class

```
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass

class Investment(Base):
```

Implementing a Model Class

```
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass

class Investment(Base):
    __tablename__ = “investment”
```

Implementing a Model Class

```
from sqlalchemy.orm import DeclarativeBase, Mapped

class Base(DeclarativeBase):
    pass

class Investment(Base):
    __tablename__ = “investment”
```

Implementing a Model Class

```
from sqlalchemy.orm import DeclarativeBase, Mapped

class Base(DeclarativeBase):
    pass

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int]
```

Implementing a Model Class

```
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
    pass

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column()
```

Implementing a Model Class

```
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
    pass

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
```

Implementing a Model Class

```
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
    pass

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
    coin: Mapped[str] = mapped_column()
    currency: Mapped[str] = mapped_column()
    amount: Mapped[float] = mapped_column()
```

Implementing a Model Class

```
from sqlalchemy import String
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
    pass

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
    coin: Mapped[str] = mapped_column(String(32))
    currency: Mapped[str] = mapped_column(String(3))
    amount: Mapped[float] = mapped_column()
```

Implementing a Model Class

```
from sqlalchemy import String, Numeric
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
    pass

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
    coin: Mapped[str] = mapped_column(String(32))
    currency: Mapped[str] = mapped_column(String(3))
    amount: Mapped[float] = mapped_column(Numeric(5, 2))
```

Model Classes and the Database

```
from sqlalchemy import create_engine
```

Model Classes and the Database

```
from sqlalchemy import create_engine

engine = create_engine(“sqlite:///data.db”)
```

Model Classes and the Database

```
from sqlalchemy import create_engine

engine = create_engine(“sqlite:///data.db”)
Base.metadata.create_all(engine)
```

Model Classes and the Database

```
from sqlalchemy import create_engine

engine = create_engine(“sqlite:///data.db”)
Base.metadata.create_all(engine)

bitcoin = Investment(coin=“bitcoin”, currency=“USD”, amount=1.0)
```

Model Classes and the Database

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///data.db”)
Base.metadata.create_all(engine)

bitcoin = Investment(coin=“bitcoin”, currency=“USD”, amount=1.0)

with Session(engine) as session:
```

Model Classes and the Database

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///data.db”)
Base.metadata.create_all(engine)

bitcoin = Investment(coin=“bitcoin”, currency=“USD”, amount=1.0)

with Session(engine) as session:
    session.add(bitcoin)
```

Model Classes and the Database

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///data.db”)
Base.metadata.create_all(engine)

bitcoin = Investment(coin=“bitcoin”, currency=“USD”, amount=1.0)

with Session(engine) as session:
    session.add(bitcoin)
    session.commit()
```

Model Classes and the Database

```
from sqlalchemy import create_engine, select
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///data.db”)
Base.metadata.create_all(engine)

bitcoin = Investment(coin=“bitcoin”, currency=“USD”, amount=1.0)

with Session(engine) as session:
    session.add(bitcoin)
    session.commit()

    stmt = select(Investment)
```

Model Classes and the Database

```
from sqlalchemy import create_engine, select
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///data.db”)
Base.metadata.create_all(engine)

bitcoin = Investment(coin=“bitcoin”, currency=“USD”, amount=1.0)

with Session(engine) as session:
    session.add(bitcoin)
    session.commit()

    stmt = select(Investment).where(Investment.coin == “bitcoin”)
```

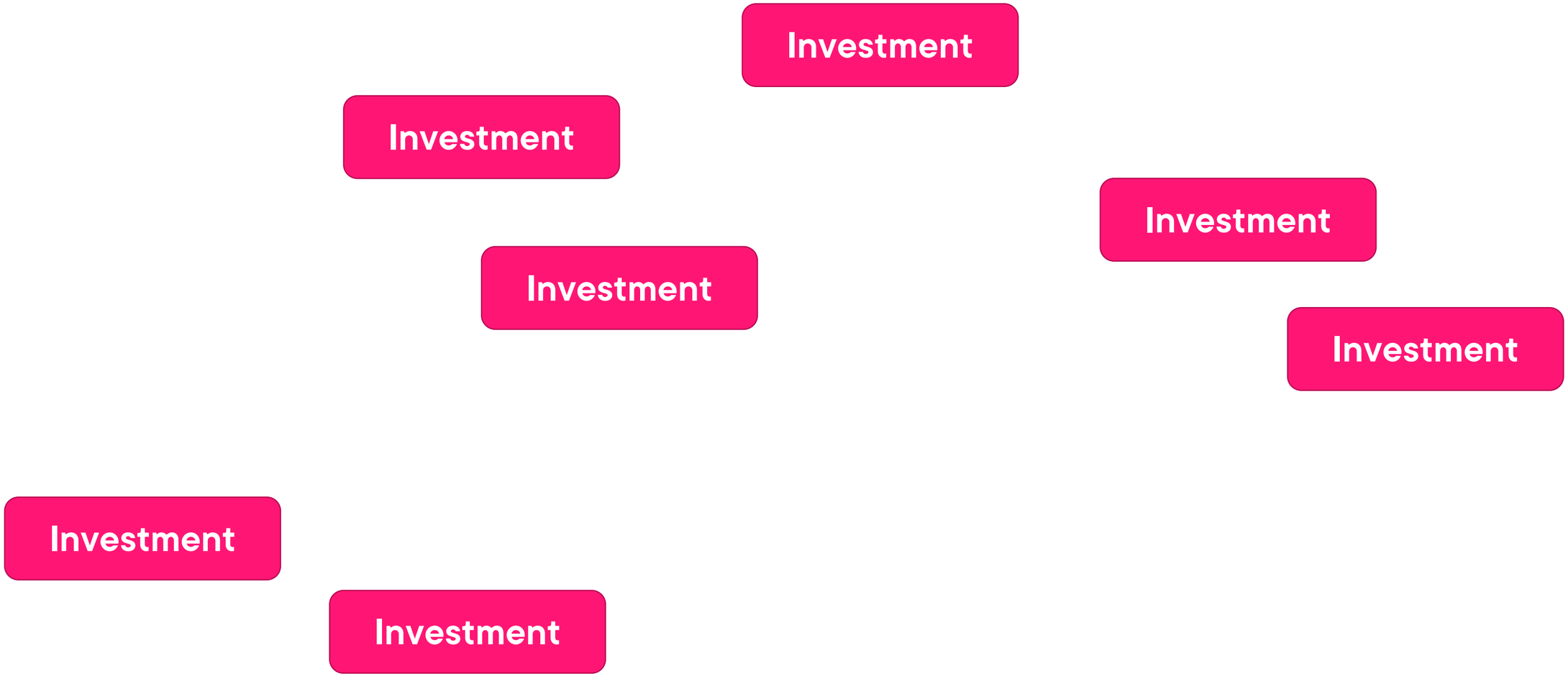
Model Classes and the Database

```
from sqlalchemy import create_engine, select
from sqlalchemy.orm import Session

engine = create_engine("sqlite:///data.db")
Base.metadata.create_all(engine)

bitcoin = Investment(coin="bitcoin", currency="USD", amount=1.0)

with Session(engine) as session:
    session.add(bitcoin)
    session.commit()

    stmt = select(Investment).where(Investment.coin == "bitcoin")
    bitcoin = session.execute(stmt).scalar_one()
```

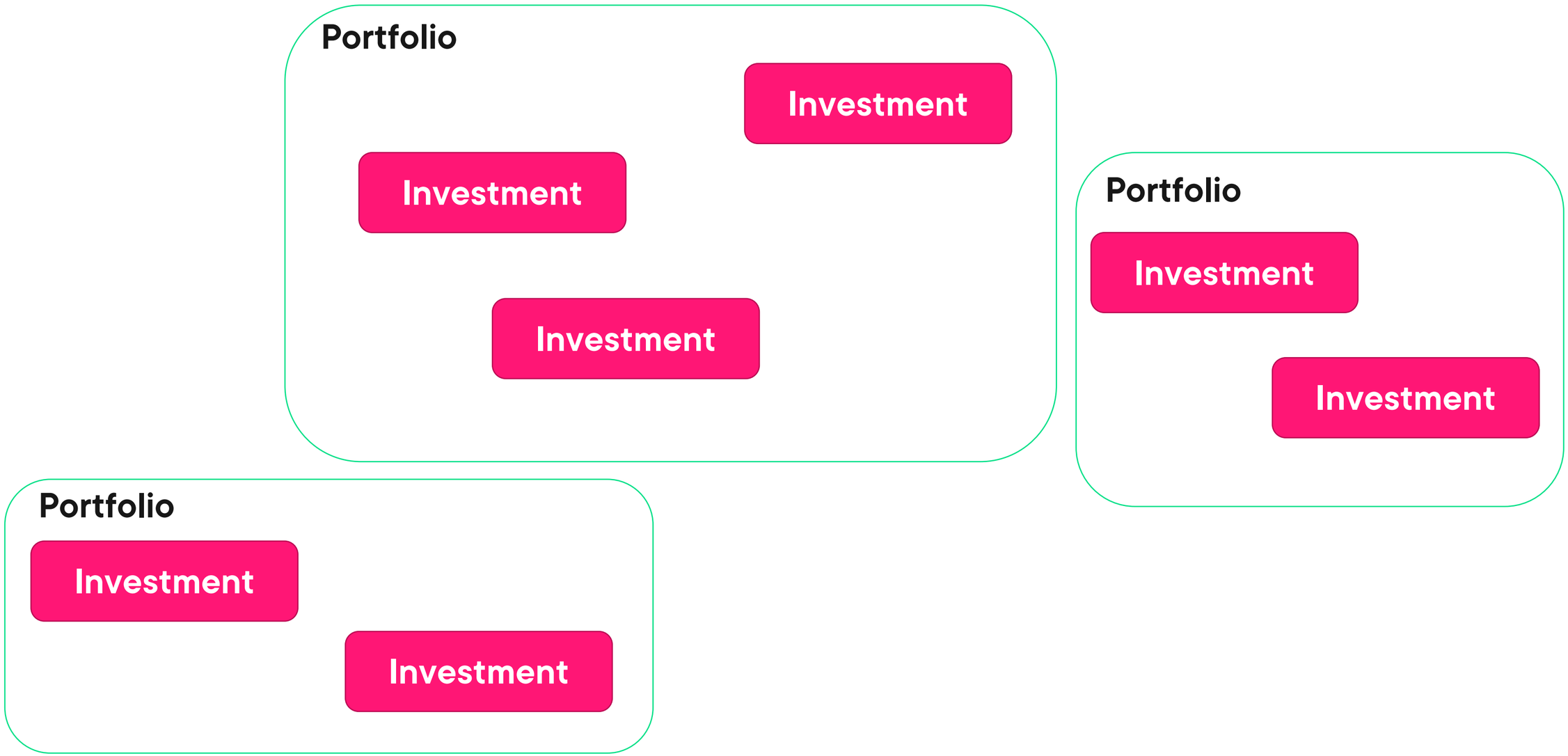
Model Classes and the Database

```
from sqlalchemy import create_engine, select
from sqlalchemy.orm import Session

engine = create_engine(“sqlite:///data.db”)
Base.metadata.create_all(engine)

bitcoin = Investment(coin=“bitcoin”, currency=“USD”, amount=1.0)

with Session(engine) as session:
    session.add(bitcoin)
    session.commit()

    stmt = select(Investment).where(Investment.coin == “bitcoin”)
    bitcoin = session.execute(stmt).scalar_one()

    print(f”{bitcoin.coin} - {bitcoin.currency} - {bitcoin.amount}”)
```

Relationships



Relationships



Portfolio Model Class

```
class Portfolio(Base):
    __tablename__ = “portfolio”
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(256))
    description: Mapped[str] = mapped_column(Text())
```

Portfolio Model Class

```
class Portfolio(Base):
    __tablename__ = “portfolio”
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(256))
    description: Mapped[str] = mapped_column(Text())

    investment_coin_1: Mapped[str] = mapped_column(String(32))
    investment_currency_1: Mapped[str] = mapped_column(String(3))
    investment_amount_1: Mapped[float] = mapped_column(Numeric(5, 2))

    investment_coin_2: Mapped[str] = mapped_column(String(32))
    investment_currency_2: Mapped[str] = mapped_column(String(3))
    investment_amount_2: Mapped[float] = mapped_column(Numeric(5, 2))
```

Pursuing a Relationship

```
class Portfolio(Base):
    __tablename__ = “portfolio”
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(256))
    description: Mapped[str] = mapped_column(Text())

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
    coin: Mapped[str] = mapped_column()
    currency: Mapped[str] = mapped_column()
    amount: Mapped[float] = mapped_column()
```

Pursuing a Relationship

```
class Portfolio(Base):
    __tablename__ = “portfolio”
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(256))
    description: Mapped[str] = mapped_column(Text())

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
    coin: Mapped[str] = mapped_column()
    currency: Mapped[str] = mapped_column()
    amount: Mapped[float] = mapped_column()
    portfolio_id: Mapped[int]
```

Pursuing a Relationship

```
class Portfolio(Base):
    __tablename__ = “portfolio”
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(256))
    description: Mapped[str] = mapped_column(Text())

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
    coin: Mapped[str] = mapped_column()
    currency: Mapped[str] = mapped_column()
    amount: Mapped[float] = mapped_column()
    portfolio_id: Mapped[int] = mapped_column(ForeignKey(“portfolio.id”))
```

Pursuing a Relationship

```
class Portfolio(Base):
    __tablename__ = “portfolio”
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(256))
    description: Mapped[str] = mapped_column(Text())

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
    coin: Mapped[str] = mapped_column()
    currency: Mapped[str] = mapped_column()
    amount: Mapped[float] = mapped_column()
    portfolio_id: Mapped[int] = mapped_column(ForeignKey(“portfolio.id”))
    portfolio: Mapped[“Portfolio”] = relationship()
```

Pursuing a Relationship

```
class Portfolio(Base):
    __tablename__ = “portfolio”
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(256))
    description: Mapped[str] = mapped_column(Text())
    investments: Mapped[List[“Investment”]] = relationship()

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
    coin: Mapped[str] = mapped_column()
    currency: Mapped[str] = mapped_column()
    amount: Mapped[float] = mapped_column()
    portfolio_id: Mapped[int] = mapped_column(ForeignKey(“portfolio.id”))
    portfolio: Mapped[“Portfolio”] = relationship()
```

Pursuing a Relationship

```
class Portfolio(Base):
    __tablename__ = “portfolio”
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(256))
    description: Mapped[str] = mapped_column(Text())
    investments: Mapped[List[“Investment”]] = relationship(back_populates=“portfolio”)

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
    coin: Mapped[str] = mapped_column()
    currency: Mapped[str] = mapped_column()
    amount: Mapped[float] = mapped_column()
    portfolio_id: Mapped[int] = mapped_column(ForeignKey(“portfolio.id”))
    portfolio: Mapped[“Portfolio”] = relationship()
```

Pursuing a Relationship

```
class Portfolio(Base):
    __tablename__ = “portfolio”
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(256))
    description: Mapped[str] = mapped_column(Text())
    investments: Mapped[List[“Investment”]] = relationship(back_populates=“portfolio”)

class Investment(Base):
    __tablename__ = “investment”
    id: Mapped[int] = mapped_column(primary_key=True)
    coin: Mapped[str] = mapped_column()
    currency: Mapped[str] = mapped_column()
    amount: Mapped[float] = mapped_column()
    portfolio_id: Mapped[int] = mapped_column(ForeignKey(“portfolio.id”))
    portfolio: Mapped[“Portfolio”] = relationship(back_populates=“investments”)
```

Adding Portfolios

```
portfolio = Portfolio(name="My Portfolio", description="My Description")

bitcoin = Investment(coin="bitcoin", currency="USD", amount=1.0)
ethereum = Investment(coin="ethereum", currency="GBP", amount=10.0)
dogecoin = Investment(coin="dogecoin", currency="EUR", amount=100.0)
```

Adding Portfolios

```
portfolio = Portfolio(name="My Portfolio", description="My Description")

bitcoin = Investment(coin="bitcoin", currency="USD", amount=1.0)
ethereum = Investment(coin="ethereum", currency="GBP", amount=10.0)
dogecoin = Investment(coin="dogecoin", currency="EUR", amount=100.0)

bitcoin.portfolio = portfolio
```

Adding Portfolios

```
portfolio = Portfolio(name="My Portfolio", description="My Description")

bitcoin = Investment(coin="bitcoin", currency="USD", amount=1.0)
ethereum = Investment(coin="ethereum", currency="GBP", amount=10.0)
dogecoin = Investment(coin="dogecoin", currency="EUR", amount=100.0)

bitcoin.portfolio = portfolio

portfolio.investments.extend([ethereum, dogecoin])
```

Adding Portfolios

```
portfolio = Portfolio(name="My Portfolio", description="My Description")

bitcoin = Investment(coin="bitcoin", currency="USD", amount=1.0)
ethereum = Investment(coin="ethereum", currency="GBP", amount=10.0)
dogecoin = Investment(coin="dogecoin", currency="EUR", amount=100.0)

bitcoin.portfolio = portfolio

portfolio.investments.extend([ethereum, dogecoin])

with Session(engine) as session:
    session.add(portfolio)
    session.commit()
```

Summary

SQLAlchemy has two APIs

- ORM
- Core API

Object Relational Mapper

Model classes

- Configure mappings to the database

Relationships

- Access related objects

Core API

- Perform CRUD tasks
- Work with typed model classes

Swap out databases

Coming up next ... NoSQL!