Relaciones entre modelos

En esta sección veremos cómo relacionar modelos con *SQLAlchemy* utilizando diversos patrones relacionales: uno a muchos, muchos a uno, uno a uno y muchos a muchos.

Relación uno a muchos

Una relación **uno a muchos** define la clave foránea en la tabla hijo referenciando a la tabla padre. Además, se puede especificar la relación en el padre también, para referenciar a una colección de ítems representado por los hijos.

```
from sqlalchemy import Table, Column, Integer, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

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")
```

Relación muchos a uno

Una relación **muchos a uno** pone la clave foránea en la tabla padre referenciando a la tabla hijo.

```
from sqlalchemy import Table, Column, Integer, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

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

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

Relación uno a uno

Una relación **uno a uno** es esencialmente bidireccional. Para lograr esto, se pone la keyword **uselist** en False para indicar que se guardará un escalar y no una lista. De esta manera, se puede transformar una relación **uno a muchos** o una relación **muchos a uno** en una relación **uno a uno**.

En el siguiente ejemplo se transforma una relación **uno a muchos** en una relación **uno a uno**.

```
from sqlalchemy import Table, Column, Integer, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

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

Relación muchos a muchos

Una relación **muchos a muchos** agrega una tabla para asociar objetos de dos modelos. La tabla de asociación se indica en el argumento **secondary** de la relación.

```
from sqlalchemy import Table, Column, Integer, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

association_table = Table('association', Base.metadata,
   Column('left_id', Integer, ForeignKey('left.id')),
   Column('right_id', Integer, ForeignKey('right.id'))
)

class Parent(Base):
   __tablename__ = 'left'
   id = Column(Integer, primary_key=True)
   children = relationship(
      "Child",
      secondary=association_table,
      back_populates="parents")

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

Ejemplo: categorías de los libros

En esta sección se muestra un ejemplo de la definición de una relación muchos a muchos. En este caso una relación que indica que un libro puede tener muchas categorías y una categoría puede tener asociado muchos libros.

```
# -*- coding: utf-8 -*-

from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String, Sequence
from sqlalchemy.orm import sessionmaker

from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship

from sqlalchemy import Table, Text

Base = declarative_base()
# association table
book_categories = Table('book_categories', Base.metadata,
   Column('book_id', ForeignKey('book.id'), primary_key=True),
   Column('category_id', ForeignKey('book_category.id'), primary_key=True)
)



class Author(Base):
   __tablename__ = 'author'

   id = Column(Integer, Sequence('author_id_seq'), primary_key=True)
```

```
    firstname = Column(String)
    lastname = Column(String)

    books = relationship("Book", order_by="Book.id", back_populates="author")

    def __repr__(self):
        return "{} {}".format(self.firstname, self.lastname)


class BookCategory(Base):
    __tablename__ = 'book_category'
    id = Column(Integer, Sequence('book_category_id_seq'), primary_key=True)
    name = Column(String)

    books = relationship('Book',
                         secondary=book_categories,
                         back_populates='categories')

    def __repr__(self):
        return "{}".format(self.name)




class Book(Base):
    __tablename__ = 'book'
    id = Column(Integer, Sequence('book_id_seq'), primary_key=True)
    isbn = Column(String)
```

```
    title = Column(String)
    description = Column(String)
    author_id = Column(Integer, ForeignKey('author.id'))

    author = relationship("Author", back_populates="books")

    categories = relationship('BookCategory',
                              secondary=book_categories,
                              back_populates='books')

    def __repr__(self):
        return "{}".format(self.title)
```

A continuación, se muestra un ejemplo de cómo instanciar y utilizar los objetos relacionados con una relación muchos a muchos.

```
>>> engine = create_engine('sqlite:///:memory:')
>>> Base.metadata.create_all(engine)

>>> Session = sessionmaker(bind=engine)
>>> session = Session()

>>> j_rowling = Author(firstname='Joanne', lastname='Rowling')

>>> session.add(j_rowling)

>>> j_rowling = session.query(Author).filter_by(firstname='Joanne').one()

>>> book = Book(isbn='9788498387087',
```

```
            title='Harry Potter y la Piedra Filosofal',
            description='La vida de Harry Potter cambia para siempre el ...')

>>> book.categories.append(BookCategory(name='Aventura'))
>>> book.categories.append(BookCategory(name='Accion'))

>>> book.author = j_rowling

>>> session.query(Book).filter(Book.categories.any(name='Aventura')).all()

>>> session.query(Book).\
            filter(Book.author==j_rowling).\
            filter(Book.categories.any(name='Aventura')).\
            all()
```