SQLAlchemy Many-to-Many

Springboard

Download Demo Code

Goals

- Make explicit joins while querying in SQLAlchemy
- Work with many-to-many relationships in SQLAlchemy

Navigating Relationships

One-to-many Navigation Review

demo/models.py

```
def phone_dir_nav():
    """Show phone dir of emps & their depts."""

    emps = Employee.query.all()

    for emp in emps:  # [<Emp>, <Emp>]
        if emp.dept is not None:
            print(emp.name, emp.dept.dept_code, emp.dept.phone)
        else:
            print(emp.name, "-", "-")
```

Joining

Can also specify joins directly

- Can be more explicit about what you want to get
- Connect tables without defined relationships
- Needed for outer joins

demo/models.py

```
def phone_dir_join():
    """Show employees with a join."""

    emps = (db.session.query(Employee.name,
                             Department.dept_name,
                             Department.phone)
            .join(Department).all())

    for name, dept, phone in emps:  # [(n, d, p), (n, d, p)]
        print(name, dept, phone)
```

You do need the ***join(cls)*** or you'll get a "cross join"

Don't forget to add `.join()` to join the second table—otherwise, you won't get an INNER JOIN, but will get a "cross join", where all employees are joined with all departments!

demo/models.py

```
def phone_dir_join_class():
    """Show employees with a join.

    This second version doesn't just get a list of data tuples,
    but a list of tuples of classes.
    """

    emps = (db.session.query(Employee, Department)
            .join(Department).all())

    for emp, dept in emps:  # [(<E>, <D>), (<E>, <D>)]
        print(emp.name, dept.dept_name, dept.phone)
```

Outer Join

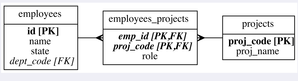
demo/models.py

```
def phone_dir_join_outerjoin():
    """Show all employees, even those without a dept."""

    emps = (db.session.query(Employee, Department)
            .outerjoin(Department).all())

    for emp, dept in emps:  # [(<E>, <D>), (<E>, <D>)]
        if dept:
            print(emp.name, dept.dept_name, dept.phone)
        else:
            print(emp.name, "-", "-")
```

Many-to-Many Relationships



employees_projects

emp_id	proj_code	role
1 (Leonard)	server	Auditor
2 (Liz)	car	Chair
2 (Liz)	server	
3 (Maggie)	car	

projects

proj_code	proj_name
car	Design Car
server	Deploy Server

Project

demo/models.py

```
class Project(db.Model):
    """Project. Employees can be assigned to this."""

    __tablename__ = "projects"

    proj_code = db.Column(db.Text, primary_key=True)
    proj_name = db.Column(db.Text,
                          nullable=False,
                          unique=True)
```

EmployeeProject

demo/models.py

```
class EmployeeProject(db.Model):
    """Mapping of an employee to a project."""

    __tablename__ = "employees_projects"

    emp_id = db.Column(db.Integer,
                       db.ForeignKey("employees.id"),
                       primary_key=True)
    proj_code = db.Column(db.Text,
                          db.ForeignKey("projects.proj_code"),
                          primary_key=True)
    role = db.Column(db.Text)
```

Relationships

demo/models.py

```
class Employee(db.Model):   # ...
    # direct navigation: emp -> employeeproject & back
    assignments = db.relationship('EmployeeProject',
                                  backref='employee')
```

demo/models.py

```
class Project(db.Model):   # ...
    # direct navigation: proj -> employeeproject & back
    assignments = db.relationship('EmployeeProject',
                                  backref='project')
```

```
>>> liz = Employee.query.get(2)
>>> liz.assignments
[<EmployeeProject 2, server>, <EmployeeProject 2, car>]

>>> car = Project.query.get('car')
>>> car.assignments
[<EmployeeProject 2, car>, <EmployeeProject 3, car>]
```

These "stop at" ***EmployeeProject***, but can go on:

```
>>> liz.assignments
[<EmployeeProject 2, server>, <EmployeeProject 2, car>]

>>> liz.assignments[0].project
<Project server Deploy Server>
```

"Through" Relationships

demo/models.py

```
class Employee(db.Model):   # ...
    # direct navigation: emp -> project & back
    projects = db.relationship('Project',
                               secondary='employees_projects',
                               backref='employees')
```

```
>>> liz.projects
<Project server Deploy Server>, <Project car Design Car>]

>>> car.employees
[<Employee 2 Liz CA>, <Employee 3 Maggie DC>]
```

These go "through" ***employees_projects*** to get result

Fine (& sometimes useful) to have both:

demo/models.py

```
class Employee(db.Model):   # ...
    # direct navigation: emp -> employeeproject & back
    assignments = db.relationship('EmployeeProject',
                                  backref='employee')

    # direct navigation: emp -> project & back
    projects = db.relationship('Project',
                               secondary='employees_projects',
                               backref='employees')
```

Adding To Relationships

Can append to "through" relationship directly:

```
>>> nadine = Employee.query.get(4)
>>> nadine.projects.append(car)
>>> db.session.commit()
>>> nadine.assignments
[<EmployeeProject 4, car>]
```

Can append to middle table:

```
>>> nadine.assignments.append(
...   EmployeeProject(proj_code='server', role='Tester'))
>>> db.session.commit()
>>> nadine.projects
[<Project server Deploy Server>, <Project car Design Car>]
```

Can add a new middle record directly:

```
>>> m_server = EmployeeProject(emp_id=3, proj_code='server')

>>> db.session.add(m_server)   # need to do this now, though
>>> db.session.commit()
```

Useful if you only have keys, not a user or project