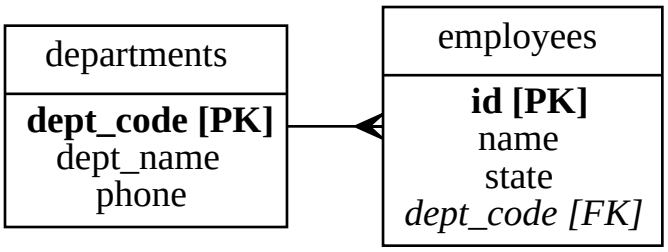
SQLAlchemy Associations

Download Demo Code <../sqla-associations-demo.zip>

Goals

- Translate relationships between tables to relationships between Python classes
- Deeper dive into SQLAlchemy querying
- Compare different approaches to querying in SQLAlchemy

Our Example



departments

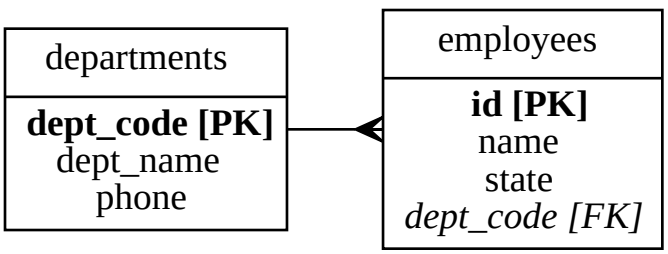
dept_code	dept_name	phone
fin	Finance	555-1000
legal	Legal	555-2222
mktg	Marketing	555-9999

employees

id	name	state	dept_code
1	Leonard	CA	legal
2	Liz	CA	legal
3	Maggie	DC	mktg
4	Nadine	CA	**null**

Relationships

Related Tables



demo/models.py

```
class Department(db.Model):
    """Department. A department has many employees."""
```

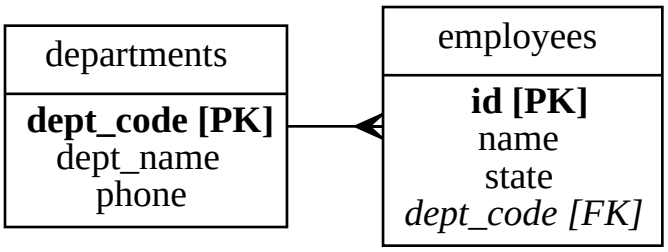
```
    __tablename__ = "departments"

    dept_code = db.Column(db.Text, primary_key=True)
    dept_name = db.Column(db.Text,
                          nullable=False,
                          unique=True)
    phone = db.Column(db.Text)
```

demo/models.py

```
class Employee(db.Model):    # ...
    dept_code = db.Column(
        db.Text,
        db.ForeignKey('departments.dept_code'))
```

- Add an actual field, ***dept_code***
- ***ForeignKey*** makes primary/foreign key relationship
  - Parameter is string “tablename.fieldname”
  - Database will handle referential integrity

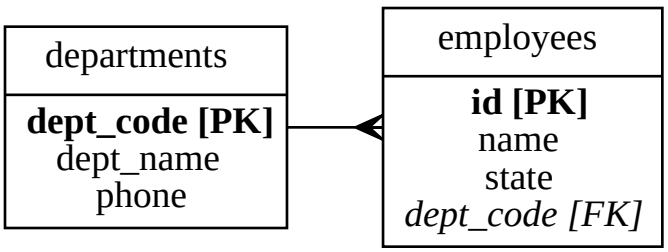


demo/models.py

```
class Employee(db.Model):    # ...
    dept_code = db.Column(
        db.Text,
        db.ForeignKey('departments.dept_code'))

    dept = db.relationship('Department')
```

- ***relationship*** allows SQLAlchemy to “navigate” this relationship
  - Using the name ***dept*** on an ***Employee***




```
class Department(db.Model):    # ...
    employees = db.relationship('Employee')
```

- Can get list of employee objects from dept with `.employees`

> **Note: Backreference**
>
> You can specify both "ends" of a database relationship as shown above: going from an employee to their department with `.dept` and from a department to their employees with `.employees`.
>
> SQLAlchemy also allows a shortcut that some people prefer—that you can just declare one relationship, and note the "backreference" for it.
>
> To do this, you wouldn't need ***.employees*** attribute on the ***Department*** class and could just put this on the ***Employee*** class:
>
> ```
> dept = db.relationship( 'Department', backref='employees')
> ```
>
> Both give the same results—you can navigate from an employee to their department and from a department to its employees, so which you use is a matter of aesthetic preference.

Navigating

```
class Employee(db.Model):    # ...
    dept = db.relationship('Department')

class Department(db.Model):  # ...
    employees = db.relationship('Employee')
```

*can navigate emp → dept with **.dept***

```
>>> leonard = Employee.query.filter_by(name='Leonard').one()

>>> leonard.dept_code
'legal'

>>> leonard.dept
<Department legal Legal>
```

*can navigate dept → emp with **.employees***

```
>>> legal = Department.query.get('legal')

>>> legal.employees
[<Employee 1 Leonard CA>, <Employee 2 Liz CA>]
```

Short-Hand Defining with Backref

longer way

```
class Employee(db.Model):    # ...
    dept = db.relationship('Department')
```

```
class Department(db.Model):  # ...
    employees = db.relationship('Employee')
```

*short-hand way using **backref***

```
class Employee(db.Model):    # ...
    dept = db.relationship('Department', backref='employees')

class Department(db.Model):  # ...
    # don't need to specify here; will auto-magically get
    # .employees to navigate to employees because of backref
```

Using Relationships

Our Goal

> *"Show phone directory of employees and their dept."*

Name	Department	Phone
Leonard	Legal	555-2222
Liz	Legal	555-2222
Maggie	Marketing	555-9999
Nadine	-	-

Navigating

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, "-", "-")
```

- Yay! So pretty! So easy!
- Not super efficient, but okay for now. (What's the problem?)

This is inefficient because SQLAlchemy fires off several queues:

- one for the list of employees
- one for *each* department

Querying

Recap

demo/models.py

```
class Employee(db.Model):
    """Employee."""

    __tablename__ = "employees"

    id = db.Column(db.Integer,
                   primary_key=True,
                   autoincrement=True)
    name = db.Column(db.Text, nullable=False, unique=True)
    state = db.Column(db.Text, nullable=False, default='CA')
```

```
SELECT * FROM employees
WHERE name = 'Liz';
```

shorter form, for simple cases

```
Employee.query.filter_by(name='Liz')
```

longer form, can use other operators

```
Employee.query.filter(Employee.name == 'Liz')
Employee.query.filter(Employee.id > 1)
```

Chaining

```
new_emps = Employee.query.filter(Employee.id > 1)

just_ca = new_emps.filter(Employee.state == 'CA')
```

Remember: nothing runs until we get results:

```
>>> just_ca
<flask_sqlalchemy.BaseQuery at 0x105234750>

>>> just_ca.all()
[<Employee 2 Liz CA>, <Employee 4 Nadine CA>]
```

More Flexible SELECT

```
SELECT * FROM employees
WHERE name = 'Liz';
```

Simple version: `ClassName.query`

```
Employee.query.filter_by(name='Liz')
Employee.query.filter(Employee.name == 'Liz')
```

More flexible version: `db.session(thing, ...).query`

```
db.session.query(Employee).filter_by(name='Liz')
db.session.query(Employee).filter(Employee.name == 'Liz')
```

This doesn't seem to gain us anything, but this general form of `db.session.query(...)` allows us to query more flexibly than a single model class.

Returning Tuples

```
SELECT id, name FROM employees;
```

```
>>> db.session.query(Employee.id, Employee.name).all()
[(1, 'Leonard'), (2, 'Liz'), (3, 'Maggie'), (4, 'Nadine')]
```

Useful when:

- You're don't need full SQLA objects
  - Don't need all fields in table
  - Don't have object to update
  - Can't call useful methods on objects
- A bit faster

Fetching Records

.all()
Get all records

.first()
Get first record, ok if there is none

.one()
Get only record, error if 0 or more than 1

.one_or_none()
Get only record, error if >1, None if 0

.count()
Get number of records found without fetching all

Get by PK

```
>>> Department.query.filter_by(dept_code='fin').one()
<Department fin Finance>
```

```
>>> Department.query.get('fin')
<Department fin Finance>

>>> Department.query.get_or_404('fin')
<Department fin Finance>
```

Common Operators

Operators

```
Employee.query.filter(Employee.name == 'Jane')

Employee.query.filter(Employee.name != 'Jane')

Employee.query.filter(Employee.id > 65)

Employee.query.filter(Employee.name.like('%Jane%'))    # LIKE

Employee.query.filter(Employee.id.in_([22, 33, 44]))   # IN ()
```

```
Employee.query.filter(Employee.state == None)      # IS NULL

Employee.query.filter(Employee.state.is_(None))    # IS NULL


Employee.query.filter(Employee.state != None)      # IS NOT NULL

Employee.query.filter(Employee.state.isnot(None)) # IS NOT NULL
```

```
q = Employee.query
```

AND:

```
q.filter(Employee.state == 'CA', Employee.id > 65)

q.filter( (Employee.state == 'CA') & (Employee.id > 65) )
```

OR:

```
q.filter( db.or_(Employee.state == 'CA', Employee.id > 65) )

q.filter( (Employee.state == 'CA') | (Employee.id > 65) )
```

NOT:

```
Employee.query.filter( db.not_(Employee.state.in_(['CA', 'OR'])) )
```

```
Employee.query.filter( ~ Employee.state.in_(['CA', 'OR']) )
```

Learning More

Self-Learning

```
q = Employee.query

q.group_by('state')

q.group_by('state').having(db.func.count(Employee.id) > 2)

q.order_by('state')

q.offset(10)

q.limit(10)
```

All described at Query Docs <http://docs.sqlalchemy.org/en/rel_1_0/orm/query.html#sqlalchemy.orm.query.Query.offset>

Learning More

SQLAlchemy Docs <http://docs.sqlalchemy.org/en/latest/>

Flask-SQLAlchemy Docs <https://pythonhosted.org/Flask-SQLAlchemy/>