The Many-to-Many Relationship

In this lesson, we will learn how to add a many-to-many relationship between models.

WE'LL COVER THE FOLLOWING

Introduction

In the last lessons, we created both a one-to-many and a one-to-one relationship between the `Employee` and `Department` models. The code so far is given below.

```
class Employee(db.Model):
    employee_id = db.Column(db.Integer, primary_key = True)
    first_name = db.Column(db.String(50), nullable = False)
    last_name = db.Column(db.String(50), nullable = False)
    department_name = db.Column(db.String, db.ForeignKey('department.name'), nullable = False
    is_head_of = db.Column(db.String, db.ForeignKey('department.name'), nullable=True)

class Department(db.Model):
    name = db.Column(db.String(50), primary_key=True, nullable=False)
    location = db.Column(db.String(120), nullable=False)
    employees = db.relationship('Employee', backref='department')
    head = db.relationship('Employee', backref='head_of_department', uselist=False)

class Project(db.Model):
    project_id = db.Column(db.Integer, primary_key=True, nullable=False)
    name = db.Column(db.String(100), nullable=False)
```

Employee, Department and Project Models

Now, let's find out how to add a *many-to-many* relationship between models.

Representing a many-to-many relationship in models

In the example, a **many-to-many** relationship should exist between the `Employee` and the `Project`. This relationship will indicate that an employee can work on multiple projects while a project can have multiple employees working on it at the same time.

Steps to add a many-to-many relationship

Creating a **many-to-many** relationship between models is totally different than the relationships we previously studied. Let's get started.

1. Create an association table called `project_members`

First, for a many-to-many relationship, we will have to create an association table. Yes, not a new model, but **a table**. This approach is recommended by the people over at `SQLAlchemy`.

```
project_members = db.Table('project_members')
```

In the snippet given above, we have created a table called `project_members`. The name of the table to be created in the database is passed to the `Table` constructor.

2. Add columns with `ForeignKey()` inside the `project_members` table

Now, we will add columns inside this table. The purpose of this table is to create an *association* between an `employee` and a `project`. Therefore, this table will have two `ForeignKey()` column: one from the `Employee` model and one from the `Project` model.

```
project_members = db.Table('project_members',
    db.Column('employee_id', db.Integer, db.ForeignKey('employee.employee_id'), primary_key=T
    db.Column('project_id', db.Integer, db.ForeignKey('project.project_id'), primary_key=True
)
```

In the snippet given above, we can find that in **lines 2 and 3**, we have added two columns and passed them to the `Table` constructor. We used the `Column`

class to create this column, like we have been doing previously. The only difference is that we passed the *name* of this column as the first parameter in the constructors. So, now we have two columns in this table: `employee_id` and `project_id`. Moreover, we set both of them as a `primary_key` because we want each entry to be **unique**.

3. Add a `relationship()` to any of the two models

Last, we need to create a way to access this association that was created by the `project_members` association table. For this purpose, we will create a `relationship()` in one of the models, i.e. `Project` or `Employee`. In this example, let's put it in the `Project` table.

```
class Project(db.Model):
    project_id = db.Column(db.Integer, primary_key=True, nullable=False)
    name = db.Column(db.String(100), nullable=False)
    members = db.relationship('Employee', secondary=project_members, backref='projects')
```

In the snippet given above, we created a `relationship()` called `members` in the `Project` model. This relationship is made with the `Employee` model. However, we used an association table to create it. Therefore, we will set the `secondary` parameter equal to the table `project_members`. And finally, we will create a `backref` in this relationship so that we can access this `project` from the `Employee` table. We named this `backref` `projects`. This means that `employee.projects` will give us all the projects that this employee is working on.

Complete implementation

In the snippet below, we can observe all the new changes we made to create the one-to-one relationship between `Employee` and `Project`.

```
class Employee(db.Model):
    employee_id = db.Column(db.Integer, primary_key = True)
    first_name = db.Column(db.String(50), nullable = False)
    last_name = db.Column(db.String(50), nullable = False)
    department_name = db.Column(db.String, db.ForeignKey('department.name'), nullable = False
    is_head_of = db.Column(db.String, db.ForeignKey('department.name'), nullable=True)

class Department(db.Model):
    name = db.Column(db.String(50), primary_key=True, nullable=False)
    location = db.Column(db.String(120), nullable=False)
    employees = db.relationship('Employee', backref='department')
    head = db.relationship('Employee', backref='head_of_department', uselist=False)
```

```
class Project(db.Model):
    project_id = db.Column(db.Integer, primary_key=True, nullable=False)
    name = db.Column(db.String(100), nullable=False)
    members = db.relationship('Employee', secondary=project_members, backref='projects')

project_members = db.Table('project_members',
    db.Column('employee_id', db.Integer, db.ForeignKey('employee.employee_id'), primary_key=T
    db.Column('project_id', db.Integer, db.ForeignKey('project.project_id'), primary_key=True
)
```

One-to-Many Relationship between Employee and Department

Well done! Now you have learned how to create relationships among models.

Now, in the next few lessons, we will be solving challenges where you will create models and relationships between models in your project. Stay tuned!