The One-to-One Relationship

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

WE'LL COVER THE FOLLOWING

Introduction

In the last lesson, we created a one-to-many 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

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')

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 learn how to add a one-to-one relationship between these models as well.

Representing a one-to-one relationship in models

In this example, a **one-to-one** relationship should exist between the `Employee` and `Department`. This relationship will indicate that an employee can be the **head** of a department (only one at a time).

Steps to add a one-to-one relationship

The steps for creating a one-to-one relationship are very similar to the steps for a one-to-many relationship with only a slight difference. So let's get started.

1. Create a column containing `ForiegnKey()`

We can add the `ForiegnKey()` column in any of the models; `Employee` or `Department`. Let's add it in `Employee` for this example.

```
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)
```

In the snippet given above, in **line 6**, we created a new `ForeignKey()` column named `is_head_of`. The `ForeignKey()` in this column is also the `department.name`.

2. Create a `relationship()` column with `backref`

Again, as we did in one-to-many, we will create a `relationship()` in the `Department` table and add a `backref` that can be used by the `employee` to access the `department` row.

```
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')
```

In the snippet given above, in **line 5**, we added a `relationship()` with the `Employee` table and a `backref` called `head_of_department`.

3. Set `useList` parameter to `False`

This step in the one-to-one relationship is what differentiates it from a one-to-many relationship. We need to make sure that this relationship is **not** built with **more than one** `employee`. Therefore, we will add an extra argument to the function called `useList` and set it to `False`. This will make sure that it does not point to a `list`.

```
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)
```

In the snippet given above, in **line 5**, we have set the `useList` parameter of the `relationship()` to `False`.

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 `Department`.

```
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)
```

One-to-Many Relationship between Employee and Department

Quick Quiz!

Q

What is the **type of the relationship** which exists between `Entity1` and `Entity2` in the snippet given below?

```
class Entity1(db.Model):
    id = db.Column(db.Integer, primary_key = True)
    column1 = db.Column(db.String, db.ForeignKey('entity2.id'
))

class Entity2(db.Model):
    id = db.Column(db.String(50), primary_key=True)
    column2 = db.relationship('Entity1', backref='column3', u
selist=True)
```

COMPLETED 0%

1 of 1

In the next lesson, we will find out how to create **many-to-many** relationships. Stay tuned!