The One-to-Many Relationship

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

WE'LL COVER THE FOLLOWING

Introduction

While working with databases, one of the key concepts is the relationship between tables. The types of relationships that exist are:

- **One-to-many**
- **One-to-one**
- **Many-to-many**

In the next few lessons, we will cover how to represent and create these relationships between the models using `SQLAlchemy`.

Consider the Human Resource Management System(HRMS) of a company. The application would contain the following models.

- `Employee`
- `Department`
- `Project`

A simple implementation of these models is given below

A simple implementation of these models 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)

class Department(db.Model):
    name = db.Column(db.String(50), nullable = False)
    location = db.Column(db.String(120), nullable = 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 learn how to add relationships between these models.

Representing a one-to-many relationship in models

This is the type of relationship we mostly encounter in real-life projects. In the example, a **one-to-many** relationship should exist between the `Employee` and the `Departement`. This relationship will indicate that an employee belongs to only one department. However, a department can contain multiple employees.

Steps to add a one-to-many relationship

1. Create a column containing `ForiegnKey()`

We will create this column inside the model. It only has one instance of the other model associated with it. Or, speaking in the context of our example, this column will be added inside the `Employee` model because an employee is only associated with **one** `Department` instance.

```
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(50), db.ForeignKey('department.name'), nullable = F
```

In the snippet given above, in **line #5**, we created a new column named

`department_name`. Moreover, we provided it with a foreign key by creating an object of the `ForeignKey` class. The constructor of this class takes the parameter `department.name`, where `department` is referring to a single row of the `Department` table; thus, it is written in lowercase. Furthermore, the `name` refers to the primary key column of the `Department` model.

2. Create a `relationship()` column

Next, we also need to make a change in the `Department` model. We will add a `relationship()` in this model to bind it with the `Employee` model.

```
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')
```

In the snippet given above, in **line #4**, we created a new column named `employees`. This column is a special type of column called a relationship. Moreover, we passed `'Employee'` as an argument to the function to specify with which model we want to create this relationship.

3. Add a `backref` argument

In the last step, we created a `relationship()` column in the `Department` model. However, we have not yet specified which rows of the `Employee` are associated with a particular row of `Department`. To cater to this problem, we will add a `backref` argument to the `relationship()` function.

```
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')
```

In the snippet given above, on **line #4**, we added a `backref` argument equal to `'department'`. What this back-reference means is that a virtual variable `department` will be created in the `Employee` table. This way, we can refer to it by `employee.department`.

> 💭 **You might wonder why we need a back-reference when we already have the `ForeignKey()` column in the `User` table?**
> Yes, we already have a foreign key. However, that only gives us the `name`

> column of the department row. The `backref` that we created in a
>
> department will act as a virtual column in the user table containing the whole row of the department.

Complete implementation

In the snippet below, we can observe all the new changes we made to create the one-to-many 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

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)
```

One-to-Many Relationship between Employee and Department

In the next lesson, we will learn how to create a one-to-one relationship in models!