Investigación sobre campos relacionales en Django

Django ofrece un mecanismo poderoso e intuitivo para ''seguir'' relaciones cuando se realizan búsquedas, haciéndose cargo de los JOINs de SQL de manera automática.

Es claro que el poder de las bases de datos se basa en relacionar tablas entre sí. Django ofrece formas de definir los tres tipos de relaciones más comunes en las bases de datos: muchos-a-uno, muchos-a-muchos, y uno-a-uno.

Relaciones de Clave Foránea (Relaciones muchos a uno)

Si un modelo contiene un ForeignKey, las instancias de ese modelo tendrán acceso al objeto relacionado (foráneo) a través de un simple atributo del modelo.

Las relaciones de clave foránea son automáticamente simétricas -- se infiere una relación inversa de la presencia de un campo ForeignKey que apunte a otro modelo.

Para definir una relación muchos-a-uno, usa un campo tipo ForeignKey. El cual se usa como cualquier otro tipo Field: incluyéndolo como un atributo de clase en tu modelo.

ForeignKey requiere un argumento posicional: la clase a la cual se relaciona el modelo. Django agrega "_id" al nombre de campo para crear su nombre de columna en la base de datos.

Por ejemplo, si un modelo Carro tiene un Fabricante -- es decir, un Fabricante fabrica múltiples carros pero cada Carro tiene solo un Fabricante -- usa la siguiente definición:

```
class Fabricante(models.Model):
    #...

class Carro(models.Model):
    fabricante = models.ForeignKey(Fabricante)
    #...
```

Para crear una relación *recursiva* – un objeto que tiene una relación muchos-a-uno consigo mismo – usa models.ForeignKey('self'):

```
class Empleado(models.Model):
    manager = models.ForeignKey('self')
```

Si necesitas crear una relación con un modelo que aún no ha sido definido, puedes usar el nombre del modelo en lugar del objeto modelo:

```
from django.db import models

class Carro(models.Model):
    fabricante = models.ForeignKey('Fabricante')
    # ...

class Fabricante(models.Model):
    # ...
```

Para referirte a modelos que han sido definidos en otra aplicación, puedes explícitamente especificar un modelo con el nombre completo de la etiqueta de la aplicación. Por ejemplo si el modelo Manufactura arriba definido es definido en otra aplicación llamada producción, necesitas usar:

```
class Carro(models.Model):
  fabricante = models.ForeignKey('produccion.Fabricante')
```

Relaciones Muchos a Muchos

Para definir una relación muchos-a-muchos, usa un campo ManyToManyField. Al igual que los campos ForeignKey, ManyToManyField requiere un argumento posicional: la clase a la cual se relaciona el modelo, trabaja de igual forma que los campos ForeignKey. Django crea una tabla join intermedia para representar la relación muchos-a-muchos.

Por ejemplo, si una Pizza tiene múltiples objetos Ingredientes -- es decir, un Ingrediente puede estar en múltiples pizzas y cada Pizza tiene múltiples ingredientes (ingredientes) -- debe representarse así:

```
class Ingredientes(models.Model):
  #...

class Pizza(models.Model):
  ingredientes = models.ManyToManyField(Ingredientes)
  ...
```

Relaciones Uno a Uno

Una relación uno-a-uno, es conceptualmente muy parecida a una relación foránea o ForeignKey con un parámetro unique=True, solo que el lado ‘‘inverso’’ de la relación devuelve directamente un único objeto.

Esto es más útil como la clave primaria de un modelo que ‘‘extiende’’ otro modelo de la misma forma; la herencia multi-tablas es implementada agregando implícitamente una relación uno a uno del modelo hijo al modelo padre.

Un argumento posicional es requerido, la clase a la cual el modelo se relaciona. Esto funciona exactamente de la misma forma en que lo hace para ForeignKey.

```
from django.conf import settings
from django.db import models

class UsuarioEspecial(models.Model):
  user = models.OneToOneField(settings.AUTH_USER_MODEL)
  supervisor = models.OneToOneField(settings.AUTH_USER_MODEL,
    related_name='supervisor')
```

Bibliografía

Garcia M., S. (2015). *La Guia definitiva de Django.* Celayita México: Django Software Corporation.