# Models

## Definicion

Un modelo corresponde a una tabla en nuestra base datos. Es la fuente de nuestros datos que se van a utilizar en la aplicación. Contiene los campos y comportamiento esenciales de los datos que se están almacenando.

## Creacion de un modelo para una ficha en Cuerpo

Para crear un modelo, agregamos un archivo vacio .py en el directorio catastro\_jnb/censo/models dentro del cual se encuentran todos los modelos del catastro. Para poder crear un modelo, hay que darse cuenta que cada modelo hereda de la clase django.db.models.Model que contiene todos los metodos necesarios para poder interactuar con la base de datos lo que permite que solo definamos los campos que contiene el modelo. A continuación, se muestra un ejemplo de como crear un modelo para una ficha de Cuerpo:

from django.db import models

class CuerpoNombreFicha(models.Model):

cuerpo = models.OneToOneField('censo.Cuerpo', blank=True)

def \_\_unicode\_\_(self):

return self.name

class Meta:

ordering = ['cuerpo']

app\_label = 'censo'

#Aqui se crea los campos especificos de cada ficha

computers\_quantity = models.IntegerField(null=True, blank=True, verbose\_name='Computadores')

observations = models.TextField(null=True, blank=True, verbose\_name='')

fk\_internet\_provider = models.ForeignKey('InternetProvider', verbose\_name='Proveedor de Internet', blank=True, null=True)

#Internal only

is\_valid = models.NullBooleanField(verbose\_name='¿Formulario válido?')

#Staff only

is\_complete = models.NullBooleanField(verbose\_name='¿Formulario completo?')

is\_correct = models.NullBooleanField(verbose\_name='¿Formulario correcto?')

## Explicación partes del modelo

### Objetos Relacionados

Cuando defines una relación en un modelo (es decir, un

ForeignKey, OneToOneField, o ManyToManyField), las instan-

cias de ese modelo tendrán una API conveniente para acceder a estos objetos relacionados. En este caso, el modelo que se esta utilizando como ejemplo esta relacionado uno a uno con el Modelo llamado Cuerpo (OneToOneField) y muchos-a-uno con el modelo InternetProvider (ForeignKey) el cual se usa como cualquier otro tipo Field: incluyéndolo como un atributo de clase en tu modelo.

Para saber más sobre relaciones, ir a la página [https://docs.djangoproject.com/en/dev/ref/models/fields/#module-django.db.models.fields.related](https://docs.djangoproject.com/en/dev/ref/models/fields/)

### Metodo \_\_unicode\_\_

Es el equivalente a \_\_str\_\_ en python, es decir, se usará cuando se quiera mostrar una instancia de la clase como una cadena de caracteres.

### class Meta:

Django usa esta class Meta interna como un lugar en el cual se pueden especificar metadatos adicionales acerca de un modelo. Es opcional.

### Metadatos: Model metadata

El modelo metadata es cualquier cosa que no sea un campo como opciones de ordenamiento, el nombre de la base de datos, etc.

### Opciones de los Metadatos del Modelo utilizadas en el ejemplo

#### ordering

El ordenamiento por omisión del objeto, utilizado cuando se obtienen listas de objetos. Esto es una tupla o lista de strings. Cada string es un nombre de campo con un prefijo opcional -, que indica orden descendiente. Los campos sin un - precedente se ordenarán en forma ascendente. Use el string a “?” para ordenar al azar.

#### app\_label

Nombre de la aplicación. En este caso es 'censo'

Para saber más de opciones de metadatos, ir a <https://docs.djangoproject.com/en/1.2/ref/models/options>.

### Creacion de atributos

Los atributos de las clases de cada modelo deberán ser instancias de unas clases que ofrece django para poder traducir estos atributos en campos de las tablas de la base de datos

Cada modelo tiene algunas variables de clase, que representan campos en la base de datos.

Cada campo está representado por una instancia de una clase models.Field -- por ejemplo, models.CharField para carácteres y models.DateTimeField para datetimes. De esta forma se le dice a Django qué tipo de datos contiene cada campo.

El nombre de cada instancia de models.Field (en este caso, computers\_quantity y observations) es el nombre del campo, en un formato amigable para la base de datos. Este valor será usado en tu código Python y tu base de datos lo usará como el nombre de la columna correspondiente.

Si quiere saber mas de tipos y opciones de campos, ir a la siguiente pagina: <https://docs.djangoproject.com/en/1.2/ref/models/fields/>

### Campos para uso interno de la administración

Los campos is\_valid, is\_correct y is\_complete son campos que pertenecen a cada ficha de cuerpo de forma obligatoria para luego ser utilizadas de forma interna por la JNB.

### Modificación de \_\_init\_\_.py

Como \_\_init\_\_.py es un archivo requerido para que Python trate a un directorio como un grupo de módulos, entonces tenemos que indicarle que se ha creado un modelo (cuerpo\_nombre\_ficha.py) agregando la siguiente linea en \_\_init\_\_.py: from cuerpo\_nombre\_ficha import CuerpoNombreFicha.

### Modelo de Material Mayor

Este modelo se diferencia de los demas, en que esta tambien relaciona al Modelo Company, es decir, se agrega la siguiente linea:

company = models.ForeignKey('censo.Company', null=True, blank=True, verbose\_name='Compañia')

## Creacion de un modelo para una ficha en compañía

Se sigue el mismo procedimiento que crear un modelo para una ficha en Cuerpo, salvo que el modelo que se crea aca esta relacionado uno-a-uno al modelo Company. En otras palabras, se agregan las siguientes lineas:

company = models.OneToOneField('censo.Company', blank=True)

def \_\_unicode\_\_(self):

return unicode(self.company)

class Meta:

ordering = ['company']

app\_label = 'censo'