Informationssysteme

Django - Webshop

Protokoll ORM

Markus Reichl

1. Mai 2017

Source

Die für den Webshop verwendete **shop/models.py** enthält folgende dokumentierte Zeilen:

```
from django.contrib.contenttypes.fields import GenericForeignKey
from django.contrib.contenttypes.models import ContentType
from django.core.exceptions import ValidationError
from django.db import models
from django.utils.datetime_safe import datetime


class Land(models.Model):
    name = models.CharField(max_length=255)
    # Liefert die Länderbezeichnung
    def __str__(self):
        return self.name


class Adresse(models.Model):
    land = models.ForeignKey(Land,on_delete=models.PROTECT)
    strasse = models.CharField(max_length=255)
    hnr = models.PositiveIntegerField()
    plz = models.CharField(max_length=20)
    ort = models.CharField(max_length=255)

    def __str__(self):
        return '{} {}, {} {} - {}'.format(
            self.strasse, self.hnr, self.plz, self.ort, self.land)
    # Mehrere Adressen bekommen einen gemeinsamen Anzeigenamen
    class Meta:
        verbose_name_plural = "Adressen"
```

```python
class Kunde(models.Model):
    name = models.CharField(max_length=255)
    passwort = models.CharField(max_length=255)
    email = models.CharField(max_length=255)
    adresse = models.ForeignKey(Adresse,on_delete=models.CASCADE)
    zuletzt_online = models.DateField(null=True)

    def __str__(self):
        return self.name


class Artikel(models.Model):
    bezeichnung = models.CharField(max_length=255)
    preis = models.DecimalField(max_digits=7, decimal_places=2)
    # Negative Stückzahlen sollen nicht möglich sein
    vstueckz = models.PositiveSmallIntegerField()
    info = models.TextField()

    def clean(self):
        """ Constraints werden in der clean Methode geprüft"""
        if self.preis <= 0:
            raise ValidationError('Preis ist negativ oder 0!')

    def __str__(self):
        return "%s, %s€, %s verfügbar - %s" % \
            (self.bezeichnung, self.preis, self.vstueckz, self.info)


class Feedback(models.Model):
    # Aufgrund der M:N Beziehung mit Kunden ist das ContentType Prinzip notwendig
    content_type = models.ForeignKey(ContentType, on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField()
    content_object = GenericForeignKey()

    kunde = models.ForeignKey(Kunde, on_delete=models.CASCADE)
    artikel = models.ForeignKey(Artikel, on_delete=models.CASCADE)
    datum = models.DateTimeField()
    bewertung = models.TextField()

    def __str__(self):
        return "%s on %s - %s | %s" % \
            (self.kunde, self.artikel.bezeichnung, self.datum, self.bewertung)
```

```
class Lieferbarkeit(models.Model):
    land = models.ForeignKey(Land, on_delete=models.CASCADE)
    artikel = models.ForeignKey(Artikel, on_delete=models.CASCADE)
    mwst = models.DecimalField(max_digits=2, decimal_places=2)

    def clean(self):
        if self.mwst <= 0:
            raise ValidationError('Mehrwertsteuer ist negativ oder 0!')

    def __str__(self):
        return "%s, %s, %s" % \
            (self.artikel.bezeichnung, self.land, self.mwst)


class Bestellung(models.Model):
    status_types = (('b', 'bestellt'), ('v', 'versendet'), ('s', 'storniert'))

    artikel = models.ManyToManyField(Artikel)
    kunde = models.ForeignKey(Kunde, on_delete=models.CASCADE)
    datum = models.DateTimeField()
    # Eine Auswahl an Werten kann durch den Parameter "choices" vorgegeben werden.
    # Dieser übernimmt ein Iterable aus Tupeln, hier "status_types".
    status = models.CharField(
        max_length=255, choices=status_types, default="bestellt")
    # Da "Adresse" mehrmals als "ForeignKey" agiert muss bei der Benutzung ein
    # "related_name" festgelegt werden, um diese zu unterscheiden.
    lieferadresse = models.ForeignKey(
        Adresse, on_delete=models.CASCADE, related_name="lads", null=False)
    rechnungsadresse = models.ForeignKey(
        Adresse, on_delete=models.CASCADE, related_name="rads", null=False)

    def clean(self):
        # Django's datetime.now() ist mit dem SQL Datentypen DATE kompatibel
        if self.datum <= datetime.now():
            raise ValidationError('Datum liegt in der Vergangenheit!')

    def __str__(self):
        return "%s, %s, %s, %s, %s, %s" % \
               (self.artikel, self.kunde, self.datum,
                self.status, self.lieferadresse, self.rechnungsadresse)


class Bestellartikel(models.Model):
    artikel = models.ForeignKey(Artikel, on_delete=models.CASCADE)
    bestellung = models.ForeignKey(Bestellung, on_delete=models.CASCADE)
    anzahl = models.PositiveSmallIntegerField()

    def __str__(self):
        return "%s, %s, %s" % (self.artikel, self.bestellung, self.anzahl)
```

```
# Die folgenden Modelle erben von Artikel und benätigen daher auch seine Referenz
class Bluray(Artikel):
    regisseur = models.CharField(max_length=255)
    jahr = models.PositiveSmallIntegerField()
    genre = models.CharField(max_length=255)
    dauer = models.IntegerField()

    def __str__(self):
        return "%s, %s, %s, %s, %s" % \
            (self.bezeichnung, self.regisseur, self.jahr, self.dauer, self.genre)
class Buch(Artikel):
    autor = models.CharField(max_length=255)
    verlag = models.CharField(max_length=255)
    seiten = models.PositiveSmallIntegerField()
    isbn = models.CharField(max_length=255)

    def __str__(self):
        return "%s, %s, %s, %s, %s" % \
        (self.bezeichnung, self.autor, self.verlag, self.isbn, self.seiten)


class SonstigerArtikel(Artikel):
    pass
```

Weiteres

Wichtige Stellen wurden bei ihrem ersten vorkommen dokumentiert, weitere Entscheidungen werden in folgendem Absatz erläutert.

Constraints

Die SQL Constraints sollten bereits vor der Speicherung in die Datenbank geprüft werden. In Django übernimmt diese Aufgabe die Methode `models.Model.clean`, welche vom Entwickler überschrieben wird. Innerhalb dieser werden dann `ValidationError`s geworfen.

Vererbung

In Django sind Paradigmen wie Vererbung und Polymorphie generell möglich, nicht anwendbar ist letzteres jedoch, wenn es sich bei der Stammklasse um eine abstrakte Klasse handelt.

ManyToMany

In Django wird die M:N Beziehung standardmäßig gesetzt. Wie man diese Einstellungen manuell vornimmt wird in https://docs.djangoproject.com/en/1.11/ref/models/fields/ erläutert.