TUTORIAL DJANGO

Model

Model merupakanrepresentasidaritabel di database. Semua attribute (field) di model akanmenjadi mapping ketabel database kita. Mapping inisudah default dari Django. Model yang akankitatambahakanmeng inherit class models Django.

from django.db import models

Cara membuat model:

1. Kita buatduluaplikasibarukitadengancara: python manage.py startapp Partner

Buka models.py yang ada di folder Partner yang baru tercreate. Kemudian tambahkan script berikut. Dalam hal ini kita akan menambah model Person

from django.db import models

# Create your models here.

class Person(models.Model):

    firstName = models.CharField(max\_length=30)

    lasttName = models.CharField(max\_length=30)

1. Setelah models.py kita update, sekarang kita akan tambahkan aplikasi baru yang kitabuat di dalam file settings.py di bagianINSTALLED\_APPS

Text

Description automatically generated

1. Kemudian kita akan melakukanmakemigrations dan migrate.

**Makemigrations** :membuatsebuah file ygberisi script untukmengcreate models ygkitabuatuntuk di migrate (dijalankanke database). Dalamhalini, kitabelummengcreatetabelke database.

**Migrate** :mengaply file migrasiygsudahtercreate di makemigrations. Dalamhalini, models kitaakan di create di database menjadisebuahtabel.

1. Setelah termigrate, kemudianuntukmenampilkankehalaman admin kitaakan register melalui file admin.py di aplikasikita.

from django.contrib import admin

from Partner.models import Person

# Register your models here.

admin.site.register(Person)

1. Berikuttampilan di halaman admin (masukmelalui: http://127.0.0.1:8000/admin/)

Graphical user interface, application, website

Description automatically generated

Fields

Fields merupakanbagianpentingdari model dan harus ada.

Contoh membuat field:

Dari contohmembuat models di atas, kitamelihatbahwa di class Person kitatelahmembuat 2 field name:

firstName = models.CharField(max\_length=30)

lasttName = models.CharField(max\_length=30)

firstNamemerupakansebuah field. Kita bisamelihattypenyaadalah Char denganmax\_lenght = 30.

**Type-type field**

1. [**choices**](https://docs.djangoproject.com/en/3.1/ref/models/fields/#django.db.models.Field.choices)

merupakan sequence tuples digunakansebagaipilihanuntuk field.

Contoh:

YEAR\_IN\_SCHOOL\_CHOICES = [

('FR', 'Freshman'),

('SO', 'Sophomore'),

('JR', 'Junior'),

('SR', 'Senior'),

('GR', 'Graduate'),

]

Element pertama : FR, SO, dllakanmenjadi value yang akandisimpanke database. Berikutcontohdalam field:

**fromdjango.dbimport**models

**classPerson**(models.Model):

SHIRT\_SIZES=(

('S','Small'),

('M','Medium'),

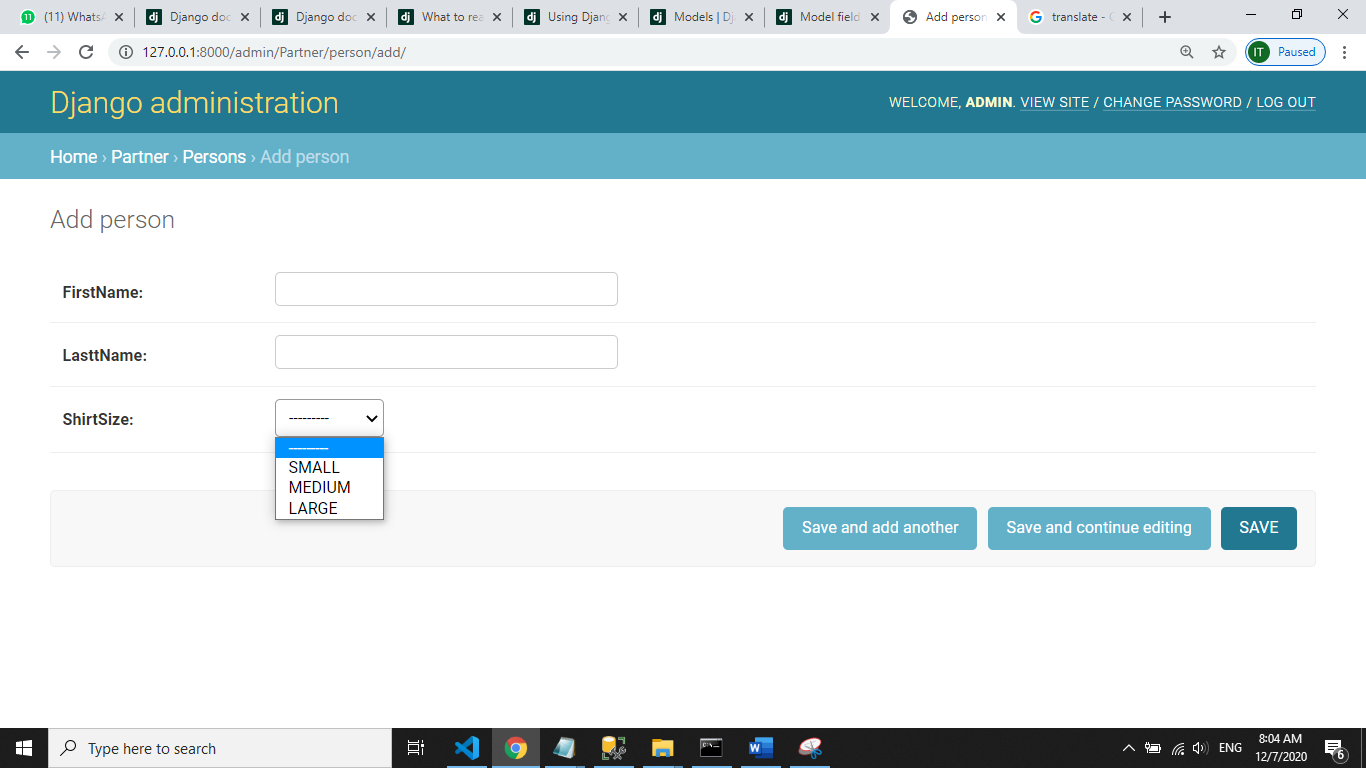
('L','Large'),

)

name=models.CharField(max\_length=60)

shirt\_size=models.CharField(max\_length=1,choices=SHIRT\_SIZES)

kemudianmakemigrations dan migrate. Berikuttampilanya:



Adapun argument optional di field Choices adalahsebagaiberikut:

[**default**](https://docs.djangoproject.com/en/3.1/ref/models/fields/#django.db.models.Field.default) : membuatnilai default untuk field tersebut. Biasanyakalaukitakosongkan pas migrate akandiingatkan

[**help\_text**](https://docs.djangoproject.com/en/3.1/ref/models/fields/#django.db.models.Field.help_text) : text bantuan

[**primary\_key**](https://docs.djangoproject.com/en/3.1/ref/models/fields/#django.db.models.Field.primary_key) : Jika True, field iniakanmenjadi primary key untuktabeltersebut. Secara default primary adalah field ID yang otomatisdibuat oleh Django.

[**unique**](https://docs.djangoproject.com/en/3.1/ref/models/fields/#django.db.models.Field.unique)

Jika True: maka field iniakanmenjadiunik (tidakbolehbernilaisama).

Nb:

Untukmerubah label field ygditampilin di view kitabisadengancarasepertiberikut:

firstName = models.CharField(“Nama pertama”,max\_length=30)

secara default nama label kitaakanmengikutinama field fieldkita.

Graphical user interface, text, application

Description automatically generated

Tapi untuk field relations: oneToOne, ManyToOne, dan ManyToMany, untukcustome label berbeda. Karena parameter pertamaadalah object relationnya. Untuk argument penamaan custom labelnya kita harus sertakan key :

verbose\_name

contoh:

sites=models.ManyToManyField(Site,verbose\_name="list of sites")

### DateField

Class aslinya:

***class*DateField(*auto\_now=False*, *auto\_now\_add=False*, *\*\*options*)**[**[source]**](https://docs.djangoproject.com/en/2.2/_modules/django/db/models/fields/#DateField)[**¶**](https://docs.djangoproject.com/en/2.2/ref/models/fields/#django.db.models.DateField)

penjelasan parameter:

**auto\_now** :merupakan attribute yang jika true, maka field date tersebut akan otomatis diisi dengan tanggal sekarang (current date). Jika field date auto\_now = true, field tersebut tidak akan bisa di override. Field akanotomatisterupdate Ketika function/ method Model.save() dipanggil.Namun jika kita ingin mengupdate data yang lain tanpa otomatis mengupdate nilai dari tanggal itu sendiri dengan mengupdate dengan querySet.

**Auto\_now\_add** :merupakan attribute yang jka true, maka field date tersebutakanotomatis juga di isi dengan tanggal sekarang. Hampirmiripdenganauto\_now, namunperbedaannya adalah nilaitanggal sekarang di isi field tersebutotomatiswaktusekarang, saatpertama row tersebut dibuat. Dengan Ketika Models.create() dipanggil.

### DateTimeField

Class asli: ***class*DateTimeField(*auto\_now=False*, *auto\_now\_add=False*, *\*\*options*)**

**Parameter samadengan*datefield***

### DecimalField

***class*DecimalField(*max\_digits=None*, *decimal\_places=None*, *\*\*options*)**

field inimemiliki 2 argument yang required, yaitu:

1. max\_digits
2. decimal\_places

contoh:

models.DecimalField(..., max\_digits=5, decimal\_places=2)

### EmailField

***class*EmailField(*max\_length=254*, *\*\*options*)**

field initermasukCharField juganamundilengkapidenganvalidasi email.

### FileField

***class*FileField(*upload\_to=None*, *max\_length=100*, *\*\*options*)**

Field yang digunakanuntukmengupload file.

Terdapat 2 attributs yang tidak required, yaitu:

1. Upload\_to :

Contoh:

**classMyModel**(models.Model):

*# file will be uploaded to MEDIA\_ROOT/uploads*

upload =models.FileField(upload\_to='uploads/')

*# or...*

*# file will be saved to MEDIA\_ROOT/uploads/2015/01/30*

upload =models.FileField(upload\_to='uploads/%Y/%m/**%d**/')

penjelasan:

sepertikitalihat di atas di field upload pertama, attribute upload\_to=’uploads/’, folder uploads ituakanberada di root yang sudahkita setting di fiel settings.py kita (MEDIA\_ROOT).

Field Relationship

#### Many-to-one relationships

Untukmembuat field many2one, kitameggunakan :  [**django.db.models.ForeignKey**](https://docs.djangoproject.com/en/3.1/ref/models/fields/#django.db.models.ForeignKey)

Foreignkeymemiliki positional argument. Maksudnyaadalahadaurutan argument yang harusdiikuti. Yaitu: untukurutanpertamaadalahnama model/class yang direlasikan.

Contoh field many2one:

**fromdjango.dbimport**models

**classManufacturer**(models.Model):

*# ...*

**pass**

**classCar**(models.Model):

manufacturer=models.ForeignKey(Manufacturer,on\_delete=models.CASCADE)

*# ...*

Dalamcontoh di atas, kitamembat field Many-to-one dari class Car ke Class Manufacturer

Untuk field Many-to-one field, kita juga bisamerelasikankediasendiri.Dengan object ‘self’.

Contoh:

**classCar**(models.Model):

manufacturer=models.ForeignKey(‘self’,on\_delete=models.CASCADE)

*# ...*

Contoh lain:

**fromdjango.dbimport**models

**classReporter**(models.Model):

first\_name=models.CharField(max\_length=30)

last\_name=models.CharField(max\_length=30)

email=models.EmailField()

**def**\_\_str\_\_(self):

**return**"**%s%s**"%(self.first\_name,self.last\_name)

**classArticle**(models.Model):

headline=models.CharField(max\_length=100)

pub\_date=models.DateField()

reporter=models.ForeignKey(Reporter,on\_delete=models.CASCADE)

**def**\_\_str\_\_(self):

**return**self.headline

**classMeta**:

ordering=['headline']

Penjelasan:

daricontoh di ataskitamelihat :

Boject**Artikel**memiliki field relasi Many-to-one ke object **Reporter**. Denga kata lain, 1 artikelhanyabisamemiliki 1 reporter, namun 1 reporter bisamemilikibanyakartikel.

Dengan field relationship itu, kitabisamengakses object Reporter (besertadatanya) dariArtikelmaupunsebaliknya.

Berikutcontohkitajalankandari python manage.py shell

Create a few Reporters:

**>>>**r=Reporter(first\_name='John',last\_name='Smith',email='john@example.com') # membuat object Report

**>>>**r.save() # menyimpan object ygbarudibuat

Create an Article:

**>>>**a=Article(id=**None**,headline="This is a test",pub\_date=date(2005,7,27),reporter=r)

**>>>**a.save()

**>>>**a.reporter.id

1

**>>>**a.reporter

<Reporter: John Smith>

Catatan:

Kita harusmenyimpan (save) object kitaitusebelumkitabisa assign data kitake object tersebut. Dengan kata lain data, tersebutharusadadidatabasedulubarubisakitapilihdia di field many-to-one kita.

**>>>**r3=Reporter(first\_name='John',last\_name='Smith',email='john@example.com')

**>>>**Article.objects.create(headline="This is a test",pub\_date=date(2005,7,27),reporter=r3)

Traceback (most recent call last):

*...*

ValueError: save() prohibited to prevent data loss due to unsaved related object 'reporter'.

Cara mengakses data reporter dari object Article:

r = a.reporter

kemudian:

r.id, r.name ,dll.

Kita bisa create object lain dari 1 object jikaada field man-to-one nya.

Contoh:

Mengcreate Article dari object Reporter:

**>>>**new\_article=r.article\_set.create(headline="John's second story",pub\_date=date(2005,7,29))

**>>>**new\_article

<Article: John's second story>

**>>>**new\_article.reporter

<Reporter: John Smith>

**>>>**new\_article.reporter.id

1

Kita tambahlagiartikelbaru:

**>>>**new\_article2=Article.objects.create(headline="Paul's story",pub\_date=date(2006,1,17),reporter=r) # r inidiambildari variable contoh di atas. Dalamhalini, python shellnya mash bersambungterurs (tidakdiclosesebelumnya)

**>>>**new\_article2.reporter

<Reporter: John Smith>

**>>>**new\_article2.reporter.id

1

**>>>**r.article\_set.all()

<QuerySet [<Article: John's second story>, <Article: Paul's story>, <Article: This is a test>]>

Cara mengaksesnya:

**>>>**r.article\_set.all()

<QuerySet [<Article: John's second story>, <Article: This is a test>]>

**>>>**r2.article\_set.all()

<QuerySet [<Article: Paul's story>]>

**>>>**r.article\_set.count()

2

**>>>**r2.article\_set.count()

1

Jika kitainginmengupdate value dari many-to-one itu, berikutcaranya:

**>>>**r2.article\_set.add(new\_article2)

**>>>**new\_article2.reporter.id

2

**>>>**new\_article2.reporter

<Reporter: Paul Jones>

Beritucaraquerysetnya :

**>>>**r.article\_set.filter(headline\_\_startswith='This')

<QuerySet [<Article: This is a test>]>

# Find all Articles for any Reporter whose first name is "John".

**>>>**Article.objects.filter(reporter\_\_first\_name='John')

<QuerySet [<Article: John's second story>, <Article: This is a test>]>

Exact match is implied here:

**>>>**Article.objects.filter(reporter\_\_first\_name='John')

<QuerySet [<Article: John's second story>, <Article: This is a test>]>

Query twice over the related field. This translates to an AND condition in the WHERE clause:

**>>>**Article.objects.filter(reporter\_\_first\_name='John',reporter\_\_last\_name='Smith')

<QuerySet [<Article: John's second story>, <Article: This is a test>]>

Untuksearch ,kitabisamasukkan primary key:

**>>>**Article.objects.filter(reporter\_\_pk=1)

<QuerySet [<Article: John's second story>, <Article: This is a test>]>

**>>>**Article.objects.filter(reporter=1)

<QuerySet [<Article: John's second story>, <Article: This is a test>]>

**>>>**Article.objects.filter(reporter=r)

<QuerySet [<Article: John's second story>, <Article: This is a test>]>

**>>>**Article.objects.filter(reporter\_\_in=[1,2]).distinct()

<QuerySet [<Article: John's second story>, <Article: Paul's story>, <Article: This is a test>]>

**>>>**Article.objects.filter(reporter\_\_in=[r,r2]).distinct()

<QuerySet [<Article: John's second story>, <Article: Paul's story>, <Article: This is a test>]>

You can also use a queryset instead of a literal list of instances:

**>>>**Article.objects.filter(reporter\_\_in=Reporter.objects.filter(first\_name='John')).distinct()

<QuerySet [<Article: John's second story>, <Article: This is a test>]>

Querying in the opposite direction:

**>>>**Reporter.objects.filter(article\_\_pk=1)

<QuerySet [<Reporter: John Smith>]>

**>>>**Reporter.objects.filter(article=1)

<QuerySet [<Reporter: John Smith>]>

**>>>**Reporter.objects.filter(article=a)

<QuerySet [<Reporter: John Smith>]>

**>>>**Reporter.objects.filter(article\_\_headline\_\_startswith='This')

<QuerySet [<Reporter: John Smith>, <Reporter: John Smith>, <Reporter: John Smith>]>

**>>>**Reporter.objects.filter(article\_\_headline\_\_startswith='This').distinct()

<QuerySet [<Reporter: John Smith>]>

Counting in the opposite direction works in conjunction with distinct():

**>>>**Reporter.objects.filter(article\_\_headline\_\_startswith='This').count()

3

**>>>**Reporter.objects.filter(article\_\_headline\_\_startswith='This').distinct().count()

1

Queries can go round in circles:

**>>>**Reporter.objects.filter(article\_\_reporter\_\_first\_name\_\_startswith='John')

<QuerySet [<Reporter: John Smith>, <Reporter: John Smith>, <Reporter: John Smith>, <Reporter: John Smith>]>

**>>>**Reporter.objects.filter(article\_\_reporter\_\_first\_name\_\_startswith='John').distinct()

<QuerySet [<Reporter: John Smith>]>

**>>>**Reporter.objects.filter(article\_\_reporter=r).distinct()

<QuerySet [<Reporter: John Smith>]>

# **Many-to-many relationships**

Cara membuat many-to-many fields, kitacukupmenambahtipefieldnyamenjadi: [**ManyToManyField**](https://docs.djangoproject.com/en/3.1/ref/models/fields/#django.db.models.ManyToManyField). Penambahannyasamaseperti field lain.

Contoh :

**fromdjango.dbimport** models

**classTopping**(models.Model):

*# ...*

**pass**

**classPizza**(models.Model):

*# ...*

toppings =models.ManyToManyField(Topping)

Penjelasan;

Dari contoh di ataskitamelihat, bahwasannya pizza memilikirelasi many-to-many dengantoppings(kumpulanpotongan). Sebab, 1 pizza memangbisamemilikibanyakpotongan, dan 1 kumpulanpotonganbisaberasaldaribeberapa pizza.

Penamanaanuntuk field denganrelasi many-to-many sebaiknyajamak (di tambah ‘s’ ). Tapitergantungpenamaanyaygkitasuka, contoh: topping\_ids, dll.

Extra many-to-many

Contoh:

Kita adaaplikasiuntukmengumpulkan group music yang dimana group music ituterdiridarisetiap orang sebagaimembernya. Oleh karenaitukitaperlumembuatrelasi many-to-many antara group denganorangnya. Namunkitaperlu juga informasimengenaikepememberandari orang tersebut. Contohtanggalbergabung di member dan lain-lain. Oleh karenaitukitaperlu juga jugamembuat 1 object Namanya membership. Nah, dari membership inilahkitamembuat 2 field sebuah field relasi man-to-one ke group dan orang.

**fromdjango.dbimport** models

**classPerson**(models.Model):

name =models.CharField(max\_length=128)

**def** \_\_str\_\_(self):

**return**self.name

**classGroup**(models.Model):

name =models.CharField(max\_length=128)

members =models.ManyToManyField(Person, through='Membership')

**def** \_\_str\_\_(self):

**return**self.name

**classMembership**(models.Model):

person =models.ForeignKey(Person, on\_delete=models.CASCADE)

group =models.ForeignKey(Group, on\_delete=models.CASCADE)

date\_joined=models.DateField()

invite\_reason=models.CharField(max\_length=64)

Penjelasan:

Dalamhalini membership menjaditabel many-to-many nya, karenakitamembuat 1 keythrough yang mengarahke model **Membership**:

through='Membership'.

Jika key tersebuttidakdibuat, maka Django akanmembuattabelbaruuntukrelasiantara groups dengan Person.

Berikutcaramenyimpan object dari python manage.py shell

**>>>**paul=Person.objects.create(name="Paul McCartney")

**>>>**beatles=Group.objects.create(name="The Beatles")

**>>>**m1 =Membership(person=ringo, group=beatles,

**...** date\_joined=date(1962, 8, 16),

**...** invite\_reason="Needed a new drummer.")

**>>>**m1.save()

**>>>**beatles.members.all()

<QuerySet [<Person: Ringo Starr>]>

**>>>**ringo.group\_set.all()

<QuerySet [<Group: The Beatles>]>

**>>>**m2 =Membership.objects.create(person=paul, group=beatles,

**...** date\_joined=date(1960, 8, 1),

**...** invite\_reason="Wanted to form a band.")

**>>>**beatles.members.all()

<QuerySet [<Person: Ringo Starr>, <Person: Paul McCartney>]>

You can also use [**add()**](https://docs.djangoproject.com/en/3.1/ref/models/relations/#django.db.models.fields.related.RelatedManager.add), [**create()**](https://docs.djangoproject.com/en/3.1/ref/models/relations/#django.db.models.fields.related.RelatedManager.create), or [**set()**](https://docs.djangoproject.com/en/3.1/ref/models/relations/#django.db.models.fields.related.RelatedManager.set) to create relationships, as long as you specify **through\_defaults** for any required fields:

**>>>**beatles.members.add(john, through\_defaults={'date\_joined': date(1960, 8, 1)})

**>>>**beatles.members.create(name="George Harrison", through\_defaults={'date\_joined': date(1960, 8, 1)})

**>>>**beatles.members.set([john, paul, ringo, george], through\_defaults={'date\_joined': date(1960, 8, 1)})

Menghapussemua value dari field many2manydengan key: clear():

The [**clear()**](https://docs.djangoproject.com/en/3.1/ref/models/relations/#django.db.models.fields.related.RelatedManager.clear) method can be used to remove all many-to-many relationships for an instance:

**>>>***# Beatles have broken up*

**>>>**beatles.members.clear()

**>>>***# Note that this deletes the intermediate model instances*

**>>>**Membership.objects.all()

<QuerySet []>

Menselect value dari many2many field;

Once you have established the many-to-many relationships, you can issue queries. Just as with normal many-to-many relationships, you can query using the attributes of the many-to-many-related model:

*# Find all the groups with a member whose name starts with 'Paul'*

>>>Group.objects.filter(members\_\_name\_\_startswith='Paul')

<QuerySet [<Group: The Beatles>]>

As you are using an intermediate model, you can also query on its attributes:

*# Find all the members of the Beatles that joined after 1 Jan 1961*

>>>Person.objects.filter(

...group\_\_name='The Beatles',

... membership\_\_date\_joined\_\_gt=date(1961,1,1))

<QuerySet [<Person: Ringo Starr]>

If you need to access a membership’s information you may do so by directly querying the **Membership** model:

**>>>**ringos\_membership=Membership.objects.get(group=beatles, person=ringo)

**>>>**ringos\_membership.date\_joined

datetime.date(1962, 8, 16)

**>>>**ringos\_membership.invite\_reason

'Needed a new drummer.'

Another way to access the same information is by querying the [many-to-many reverse relationship](https://docs.djangoproject.com/en/3.1/topics/db/queries/#m2m-reverse-relationships) from a **Person** object:

**>>>**ringos\_membership=ringo.membership\_set.get(group=beatles)

**>>>**ringos\_membership.date\_joined

datetime.date(1962, 8, 16)

**>>>**ringos\_membership.invite\_reason

'Needed a new drummer.'

Membuat field relation, tapimodelnyaada di aplikasilain.

Caranya:

Import model yang inginkitarelasikan, kemudianmasukanyg di import menjadiargmentpertamadifieldkita. Dimana biasanya argument pertamaadalah object dari yang kitarelasikan.

Contoh:

**fromdjango.dbimport** models

**fromgeography.modelsimport**ZipCode

**classRestaurant**(models.Model):

*# ...*

zip\_code=models.ForeignKey(

ZipCode,

on\_delete=models.SET\_NULL,

blank=**True**,

null=**True**,

)

Cara custom penamaan table di Django supayatidakmengikuti default:

Di modelnyakitakasihsubclass : class meta. Kemudiankasih Key: db\_table: nama\_table\_yg\_diinginkan.

Contoh:

Class Album(models.Model)

Class meta:

db\_table='music\_album'

Manager

Untukqueryset, kitaakangunakannamaObject.**objects.**all()

Dalamhalini Manager adalah**objects.** Kita bisamenambahfungsikitasendirikedalam manager inidenganmengiheritdari class:

models.Manager

contoh:

**fromdjango.dbimport** models

**classPollManager**(models.Manager):

**def**with\_counts(self):

**fromdjango.dbimport** connection

**with**connection.cursor() **as** cursor:

cursor.execute("""

SELECT p.id, p.question, p.poll\_date, COUNT(\*)

FROM polls\_opinionpoll p, polls\_response r

WHERE p.id = r.poll\_id

GROUP BY p.id, p.question, p.poll\_date

ORDER BY p.poll\_date DESC""")

result\_list= []

**for** row **in**cursor.fetchall():

p =self.model(id=row[0], question=row[1], poll\_date=row[2])

p.num\_responses=row[3]

result\_list.append(p)

**return**result\_list

**classOpinionPoll**(models.Model):

question =models.CharField(max\_length=200)

poll\_date=models.DateField()

objects =PollManager()

**classResponse**(models.Model):

poll =models.ForeignKey(OpinionPoll, on\_delete=models.CASCADE)

person\_name=models.CharField(max\_length=50)

response =models.TextField()

## **Model methods**

caramenambah method atau function kedalam object/class kitaadalahseperticontohberikut:

**fromdjango.dbimport** models

**classPerson**(models.Model):

first\_name=models.CharField(max\_length=50)

last\_name=models.CharField(max\_length=50)

birth\_date=models.DateField()

**def**baby\_boomer\_status(self):

"Returns the person's baby-boomer status."

**importdatetime**

**if**self.birth\_date<datetime.date(1945, 8, 1):

**return**"Pre-boomer"

**elif**self.birth\_date<datetime.date(1965, 1, 1):

**return**"Baby boomer"

**else**:

**return**"Post-boomer"

@property

**def**full\_name(self):

"Returns the person's full name."

**return**'**%s%s**'% (self.first\_name, self.last\_name)

caramengoverride method bawaannya Django

1. Save

**fromdjango.dbimport** models

**classBlog**(models.Model):

name =models.CharField(max\_length=100)

tagline =models.TextField()

**def**save(self, \*args, \*\*kwargs):

do\_something()

super().save(\*args, \*\*kwargs) *# Call the "real" save() method.*

do\_something\_else()

sebenarnyamasihbanyak method default Django. Seperti update, delete, dll.

Model inheritance

Inherit dari class abstract

###########

### Model Abstract

###########

class CommonInfo(models.Model):

    name = models.CharField(max\_length=100)

    age = models.PositiveIntegerField()

    class Meta:

        abstract = True

class Student(CommonInfo):

    home\_group = models.CharField(max\_length=5)

perhatikanuntukmembuat model abstract, seperti yang kitalihat di class CommonInfoadalahdengancaramenambah abstract = True di subclass Meta nya.

Class abstract itutidakakan di create database walaupunkitasudah migrate. Tujuan class abstract ituadalahuntukdipakai di model lain. Contoh di Student ituhanyaada field home\_group. Secara default jikakitatidak inherit ke class abstract, Ketika migrate, tabelituhanyamemiliki field home\_group di tabelnya. Namun Ketika inherit, tabel Student akanmemiliki juga fied-field induk class, yaitu name, age.

Fields

Field di model Django merupakan instantiasi dari sebuah class

Cara menginherit sebuah meta class class parent:

Contoh:

**from** **django.db** **import** models

**class** **CommonInfo**(models.Model):

*# ...*

**class** **Meta**:

abstract = **True**

ordering = ['name']

**class** **Student**(CommonInfo):

*# ...*

**class** **Meta**(CommonInfo.Meta):

db\_table = 'student\_info'

Model

inheritance

Ada 3 jenis cara inheritance model di Django:

1. Abstract base class

2. Multi-table inheritance : sebuah class bisa meng-inherit lebih dari 1 parent class

3. Proxy : jika kita hanya ingin mengubah behavior dari parent class, tidak ada penambahan field. Dengan kata lain hanya menambah function.

Contoh:

1. Abstract base class

Base/parent class nya , digunakan hanya untuk pemodelan, tanpa implementasi

Contoh:

**rom** **django.db** **import** models

**class** **CommonInfo**(models.Model):

name = models.CharField(max\_length=100)

age = models.PositiveIntegerField()

**class** **Meta**:

abstract = **True**

**class** **Student**(CommonInfo):

home\_group = models.CharField(max\_length=5)

2.Multi-table inheritance

Sebuah class bisa meng-inherit lebih dari 1 parent class

Contoh:

**from** **django.db** **import** models

**class** **Place**(models.Model):

name = models.CharField(max\_length=50)

address = models.CharField(max\_length=80)

**class** **Restaurant**(Place):

serves\_hot\_dogs = models.BooleanField(default=**False**)

serves\_pizza = models.BooleanField(default=**False**)

**cara menyimpan foreign key:**

untuk menyimpan ini, cara menyimpan sama seperti menyimpan field biasa. tapi untuk field yang akan di simpan adalah berupa instatiate. Perhatikan contoh berikut:

**>>> from** **blog.models** **import** Blog, Entry

**>>>** entry = Entry.objects.get(pk=1)

**>>>** cheese\_blog = Blog.objects.get(name="Cheddar Talk")

**>>>** entry.blog = cheese\_blog

**>>>** entry.save()

Perhatikan baris ke-3 , disana kita lihat bahwa. Kita ambil dulu parent foreign key yang mau di simpan ke table entry. Setelah itu baru bisa kita simpan(save).

**Cara menyimpan many-to-many field**

Untuk menyimpan ini, ada sedikit perbedaan cara. Untuk field type many-to-many, kita harus gunakan fungsi **save()**.

Contoh:

**>>> from** **blog.models** **import** Author

**>>>** joe = Author.objects.create(name="Joe")

**>>>** entry.authors.add(joe)

Untuk menyimpan banyak value dari many-to-many field sekaligus, kita dapat melakukan dengan cara seperti berikut:

**>>>** john = Author.objects.create(name="John")

**>>>** paul = Author.objects.create(name="Paul")

**>>>** george = Author.objects.create(name="George")

**>>>** ringo = Author.objects.create(name="Ringo")

**>>>** entry.authors.add(john, paul, george, ringo)

Penjelasan:

Dari contoh di atas, kita lihat bahwasannya yang akan di assign itu, kita define dulu. Kemudian dibagian paling bawah, kita add ke object tujuan kita.

Signals

Signals digunakan untuk men triger action lain dijalankan ketika ada 1 aksi dilakukan. Berikut beberapa signals yang sudah disediakan oleh Django (set of built signals):

1. [**django.db.models.signals.pre\_save**](https://docs.djangoproject.com/en/3.1/ref/signals/#django.db.models.signals.pre_save) & [**django.db.models.signals.post\_save**](https://docs.djangoproject.com/en/3.1/ref/signals/#django.db.models.signals.post_save)

Ini adalah dipanggil ketika sebelum dan sesudah method **save** dipanggil di suatu model.

2. [**django.db.models.signals.pre\_delete**](https://docs.djangoproject.com/en/3.1/ref/signals/#django.db.models.signals.pre_delete) & [**django.db.models.signals.post\_delete**](https://docs.djangoproject.com/en/3.1/ref/signals/#django.db.models.signals.post_delete)

Ini adalah dipanggil ketika sebelum dan sesudah method save **delete** dipanggil di suatu model.

3. [**django.db.models.signals.m2m\_changed**](https://docs.djangoproject.com/en/3.1/ref/signals/#django.db.models.signals.m2m_changed)

Dipanggil ketika sebuah field many2many field diupdate atau berubah.

[4. **django.core.signals.request\_started**](https://docs.djangoproject.com/en/3.1/ref/signals/#django.core.signals.request_started) & [**django.core.signals.request\_finished**](https://docs.djangoproject.com/en/3.1/ref/signals/#django.core.signals.request_finished)

Dipanggil ketika sebuah request mulai dan sesudah dijalankan. Ini berlaku di **request http**

Contoh untuk mengcreate profile ketika sebuah user ditambah. Dalam hal ini flownya adalah, ketika kita create sebuah user, maka dengan otomatis 1 data profile juga akan bertambah. Perhatikan contoh berikut:

from django.db import models

from django.contrib.auth.models import User

from django.db.models.signals import post\_save

from django.dispatch import receiver

class MyUser(User):

    class Meta:

        proxy = True

def create\_profile(sender, instance, created, \*\*kwargs):

    if created:

        Profile.objects.create(user=instance)

        print ('profile created')

post\_save.connect(create\_profile, sender=MyUser)

class Profile(models.Model):

    user = models.OneToOneField(MyUser, on\_delete=models.CASCADE, blank=True, null=True)

    first\_nama = models.CharField(max\_length=70, blank=True)

penjelasan:

pertama perhatikan pada script ini:

def create\_profile(sender, instance, created, \*\*kwargs):

    if created:

        Profile.objects.create(user=instance)

        print ('profile created')

post\_save.connect(create\_profile, sender=MyUser)

disana kita membuat sebuah function yang berada di luar class. Namanya adalah create\_profile(sender, instance, created, \*\*kwargs)

sender: object yang mentrigger

instance : nilai dari object yang akan disimpan

created : true/false, menyatakan apakah aksi/trigger nya adalah menambah data atau hanya mengupdate saja.

kemudian kita panggil function tersebut dengan script:

post\_save.connect(create\_profile, sender=MyUser)

post\_save itu merupakan function yang kita inherit dari **signals** milik Django.

from django.db.models.signals import post\_save