Django Models

**Summary**: in this tutorial, you’ll learn about Django models and how to create models for your Django application.

# Introduction to Django models

In Django, a model is a subclass of the django.db.models.Model class. A model contains one or more fields and methods that manipulate the fields.

Essentially, a Django model maps to a single table in the database in which each field of the model represents a column in the table.

An application may have zero or more models stored in models.py module. For example, the following defines a Post model for the blog application:

from django.db import models

from django.utils import timezone

class Post(models.Model):

title = models.CharField(max\_length=120) content = models.TextField()

published\_at = models.DateTimeField(default=timezone.now)

The Post model has the title , content , and published\_at fields. Based on the Post

model, Django will create a table in the database with the following SQL code

CREATE TABLE "blog\_post" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"title" varchar(120) NOT NULL, "content" text NOT NULL,

"published\_at" datetime NOT NULL,

);

Note that the above-generated [SQL](https://www.sqltutorial.org/) is for [SQLite](https://www.sqlitetutorial.net/). If you use a different database, you’ll see that the SQL code is slightly different.

The name of the table blog\_post is automatically derived from the application and model names:

application.model

In this example, Django will create a table blog\_post for the Post model.

To specify a table name instead of using the default name generated by Django, you can use the

db\_table attribute of the Meta class like this:

from django.db import models

from django.utils import timezone

class Post(models.Model):

title = models.CharField(max\_length=120) content = models.TextField()

published\_at = models.DateTimeField(default=timezone.now)

class Meta:

db\_table = 'posts'

In this case, the Post model will map to the posts table instead of the generated blog\_post table. In this tutorial, we’ll sticky with the default generated table name blog\_post .

When creating a table, Django automatically adds the id field as the primary key of the table. The id field is an auto-increment field with the type specified in the settings.py file of the project:

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

If you want to specify your own primary key field, you need to explicitly define it in the model like this:

post\_id = models.BigAutoField(primary\_key=True)

In this example, the primary\_key=True indicates that the post\_id is a [primary key](https://www.sqlitetutorial.net/sqlite-primary-key/). When Django sees a field in the model with the primary\_key=True , it won’t add the automatic id column.

Django requires each model to have **exactly one field** with the primary\_key=True .

## Using models

Once defining models, you need to tell Django that you’re going to use them by registering the application name in the INSTALLED\_APPS list in the settings.py of the project:

INSTALLED\_APPS = [

*# ...*

'blog.apps.BlogConfig',

]

## Built-in models

Django comes with some built-in models like User from django.contrib.auth.models

module. To use the User model, you need to import it into the models.py file:

from django.contrib.auth.models import User

## Foreign keys

Each post in the blog application is created by a user and a user may create zero or more posts. This is called a [one-to-many relationship](https://www.pythontutorial.net/django-tutorial/django-one-to-many/).

To model the one-to-many relationship, you use the ForeignKey field:

from django.db import models

from django.utils import timezone

from django.contrib.auth.models import User

class Post(models.Model):

title = models.CharField(max\_length=120) content = models.TextField()

published\_at = models.DateTimeField(default=timezone.now)

author = models.ForeignKey(User, on\_delete=models.CASCADE)

Based on this model, Django will create the blog\_post table with the following structure:

CREATE TABLE "blog\_post" (

"id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"title" varchar(120) NOT NULL, "content" text NOT NULL, "published\_at" datetime NOT NULL, "author\_id" integer NOT NULL

REFERENCES "auth\_user" ("id") DEFERRABLE INITIALLY DEFERRED

);

In this example, the auth\_id is a [foreign key](https://www.sqlitetutorial.net/sqlite-foreign-key/) that creates a relationship between blog\_post

table and auth\_user table. Note that the auth\_user table is the Django-provided table.

## The str () method

To define the string representation of a model, you can override the str () method. For example:

from django.db import models

from django.utils import timezone

from django.contrib.auth.models import User

class Post(models.Model):

title = models.CharField(max\_length=120) content = models.TextField()

published\_at = models.DateTimeField(default=timezone.now)

author = models.ForeignKey(User, on\_delete=models.CASCADE)

def str (self):

return self.title

When you use the instance of the Post model as a string, Django calls the  [str ()](https://www.pythontutorial.net/python-oop/python-__str__/) method and displays its result.

## Adding Meta class to the Model class

The Meta class allows you to configure the model. For example, the following defines the

Meta class inside the Post model class that sorts the posts by the published\_at in descending order ( -published\_at ) i.e., the newer posts first and the older posts after.

from django.db import models

from django.utils import timezone

from django.contrib.auth.models import User

class Post(models.Model):

title = models.CharField(max\_length=120) content = models.TextField()

published\_at = models.DateTimeField(default=timezone.now)

author = models.ForeignKey(User, on\_delete=models.CASCADE)

def str (self):

return self.title

class Meta:

ordering = ['-published\_at']

After defining models, you can create and apply migrations to create tables in the database, which we’ll cover in the next tutorial.

# Summary

Define all models in the models.py file of the Django application.

Define a class that inherits from the django.db.models.Model to create a model.

A model maps to a table in the database, in which each field maps to a column in the database table.

Override str () method to return a string representation of a model. Use the Meta class to configure the model.