Django Migrations

**Summary**: in this tutorial, you’ll learn how to create models and use Django migrations to create database tables.

# Introduction to Django migration commands

When working with Django, you don’t need to write SQL to create new tables or make changes to existing tables. Instead, you use Django migrations.

Django migrations allow you to propagate the changes that you make to the [models](https://www.pythontutorial.net/django-tutorial/django-models/) to the database via the command line.

Django provides you with some commands for creating new migrations based on the changes that you made to the model and applying the migrations to the database.

The process for making changes to models, creating migrations, and applying the changes to the database is as follows:

First, define new models or make changes to existing models.

Second, make new migrations by running the makemigrations command.

Third, apply the changes from the models to the database by executing the migrate

command.

Suppose that you define the Post models in the blog application like this:

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

and you can create a new migration using the makemigrations command:

python manage.py makemigrations

The makemigrations command scans the models.py file, detects changes, and makes corresponding migrations. It’ll show the following output:

Migrations for 'blog': blog\migrations\0001\_initial.py

- Create model Post

Behind the scene, the command creates the file migrations\0001\_initial.py file.

To preview the SQL that Django will run to create the blog\_post table in the database, you use the sqlmigrate command:

python manage.py sqlmigrate blog 0001

In this sqlmigrate command, the blog is the name of the application and 0001 is the migration number.

It’ll output the following:

BEGIN;

*--*

*-- Create model Post*

*--*

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

);

CREATE INDEX "blog\_post\_author\_id\_dd7a8485"

ON "blog\_post" ("author\_id");

COMMIT;

To apply the changes to the database, you execute the migrate command:

python manage.py migrate

It’ll show the following output:

Operations to perform:

Apply all migrations: admin, auth, blog, contenttypes, sessions Running migrations:

Applying contenttypes.0001\_initial... OK

Applying auth.0001\_initial... OK Applying admin.0001\_initial... OK

Applying admin.0002\_logentry\_remove\_auto\_add... OK

Applying admin.0003\_logentry\_add\_action\_flag\_choices... OK Applying contenttypes.0002\_remove\_content\_type\_name... OK Applying auth.0002\_alter\_permission\_name\_max\_length... OK Applying auth.0003\_alter\_user\_email\_max\_length... OK Applying auth.0004\_alter\_user\_username\_opts... OK

Applying auth.0005\_alter\_user\_last\_login\_null... OK

Applying auth.0006\_require\_contenttypes\_0002... OK

Applying auth.0007\_alter\_validators\_add\_error\_messages... OK Applying auth.0008\_alter\_user\_username\_max\_length... OK Applying auth.0009\_alter\_user\_last\_name\_max\_length... OK

Applying auth.0010\_alter\_group\_name\_max\_length... OK Applying auth.0011\_update\_proxy\_permissions... OK Applying auth.0012\_alter\_user\_first\_name\_max\_length... OK Applying blog.0001\_initial... OK

Applying sessions.0001\_initial... OK

Note that besides applying the migration for the Post model, Django also applied the migrations for the built-in models used in authentication, authorization, sessions, etc.

If you execute the migrate command again and there are no unapplied migrations, the command will output the following:

Operations to perform:

Apply all migrations: admin, auth, blog, contenttypes, sessions, users Running migrations:

No migrations to apply.

To list the project migrations and their status, you use the showmigrations command:

python manage.py showmigrations

Output:

admin

[X] 0001\_initial

[X] 0002\_logentry\_remove\_auto\_add

[X] 0003\_logentry\_add\_action\_flag\_choices auth

[X] 0001\_initial

[X] 0002\_alter\_permission\_name\_max\_length

[X] 0003\_alter\_user\_email\_max\_length

[X] 0004\_alter\_user\_username\_opts

[X] 0005\_alter\_user\_last\_login\_null

[X] 0006\_require\_contenttypes\_0002

[X] 0007\_alter\_validators\_add\_error\_messages

[X] 0008\_alter\_user\_username\_max\_length

[X] 0009\_alter\_user\_last\_name\_max\_length

[X] 0010\_alter\_group\_name\_max\_length

[X] 0011\_update\_proxy\_permissions

[X] 0012\_alter\_user\_first\_name\_max\_length blog

[X] 0001\_initial contenttypes

[X] 0001\_initial

[X] 0002\_remove\_content\_type\_name sessions

[X] 0001\_initial

# Summary

Use the makemigrations command to make migrations based on the changes that you made to the models.

Use the migrate command to apply changes from models to the database.

Use the sqlmigrate command to view the generated SQL based on the model.

Use the showmigrations command to list all migrations and their status in the project.