**Model Form in Django**

**5 code file ch35 video 41**

* Mirrors your code (models, forms, views, templates).
* Shows labels, widgets, attrs, error messages.
* Shows **overriding model fields in the form** (e.g., max\_length=50) and what that means.
* Adds an **extra field not in the model** (conform\_passward) and validates it.
* Explains **save / save(commit=False)** correctly.
* Covers **create, update (two ways), delete**, fields='\_\_all\_\_' vs exclude, blank=True, and how model fields map to form fields.

# 1) models.py (your model)

*# student/models.py*

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

*#* **NOTE***: 'passward' is kept as-is to match your code everywhere.*

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

    name **=** models**.**CharField(**max\_length=**255**,** **blank=**True)  *# blank=True -> NOT required in forms*

    email **=** models**.**EmailField(**max\_length=**255)

    passward **=** models**.**CharField(**max\_length=**255)

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

*# helps in admin and shell*

**return** f"{*self***.**name **or** '(no name)'} <{*self***.**email}>"

**Key points**

* blank=True on name means: in **forms**, required=False. In the DB it can still be empty string; it does not automatically set NULL (that’s null=True, for non-text fields typically).
* Model → Form auto mapping (Django 5):
  + ForeignKey → ModelChoiceField
  + ManyToManyField → ModelMultipleChoiceField
  + BooleanField(null=True) → rendered by ModelForm using a **three-state** widget (like NullBooleanSelect)
  + AutoField/BigAutoField (primary key) are **excluded** by default (they’re not editable).
  + Fields with editable=False are excluded.
  + blank=True → field is **not required** in the form.

⚠️ NullBooleanField model field was removed in Django 4+. Use BooleanField(null=True) on the model. The **form** still has NullBooleanField, and a ModelForm will pick the right widget if your boolean can be null.

# 2) admin.py

*# student/admin.py*

**from** django**.**contrib **import** admin

**from** student**.**models **import** Profile

@admin**.**register(Profile)

class **ProfileAdmin**(*admin***.***ModelAdmin*)**:**  *# name this ProfileAdmin to avoid confusion with model class*

    list\_display **=** ('id'**,** 'name'**,** 'email'**,** 'passward')

# 3) Basic ModelForm (your starter)

*# student/forms.py*

**from** django **import** forms

**from** student**.**models **import** Profile

class **RegistrationForm**(*forms***.***ModelForm*)**:**

    class **Meta:**

        model **=** Profile

*#use fileds one by one or use \_\_all\_\_*

        fields **=** ['name'**,** 'email'**,** 'passward']  *# or '\_\_all\_\_'*

*#use labels like this*

        labels **=** {

            'name'**:** 'ENTER NAME'**,**

            'email'**:** 'Enter Email:'**,**

            'passward'**:** 'Enter Passward'**,**

        }

        error\_messages **=** {

*# These attach to model fields; keys are field names, values are dict of error\_code->message*

            'email'**:** {'required'**:** 'Email is required'}

        }

        widgets **=** {

*# IMPORTANT: call the widget class (add parentheses) if you’re passing attrs*

            'passward'**:** forms**.**PasswordInput()**,**  *# widget instance (no attrs yet)*

        }

**Notes**

* Meta.labels, Meta.error\_messages, Meta.widgets apply to **all fields**, including ones you **override** in the form (unless you explicitly set a label/errors on the override itself).
* If you need attrs (CSS classes/placeholders), pass a widget instance with attrs.

# 4) Adding attrs to widgets

class **RegistrationForm**(*forms***.***ModelForm*)**:**

    class **Meta:**

        model **=** Profile

        fields **=** ['name'**,** 'email'**,** 'passward']

        labels **=** {

            'name'**:** 'ENTER NAME'**,**

            'email'**:** 'Enter Email:'**,**

            'passward'**:** 'Enter Passward'**,**

        }

        error\_messages **=** {

            'email'**:** {'required'**:** 'Email is required'}

        }

#here are widget code

        widgets **=** {

            'passward'**:** forms**.**PasswordInput(**attrs=**{'class'**:** 'pwdclass'})**,**

            'name'**:** forms**.**TextInput(**attrs=**{'class'**:** 'nameclass'**,** 'placeholder'**:** 'enter your name '})

        }

# 5) Overriding a ****model field**** in the form (e.g., change max\_length)

You said: “We set max\_length=255 in the model, but I want 50 in the form.”  
✅ This is fine: the **form will validate up to 50**, but the DB column is still 255. (So the DB can store more, but users can only submit up to 50 through this form.)

class **RegistrationForm**(*forms***.***ModelForm*)**:**

*# OVERRIDES the model's generated form field*

    name **=** forms**.**CharField(

**max\_length=**50**,**                       *# stricter than model*

*# If you DON'T set label/error\_messages here,*

*# Meta.labels / Meta.error\_messages will still apply.*

**widget=**forms**.**TextInput(**attrs=**{

            'class'**:** 'nameclass'**,**

            'placeholder'**:** 'enter your name '

        })

    )

    class **Meta:**

        model **=** Profile

        fields **=** ['name'**,** 'email'**,** 'passward']

        labels **=** {'name'**:** 'ENTER NAME'**,** 'email'**:** 'Enter Email:'**,** 'passward'**:** 'Enter Passward'}

        error\_messages **=** {'email'**:** {'required'**:** 'Email is required'}}

        widgets **=** {'passward'**:** forms**.**PasswordInput(**attrs=**{'class'**:** 'pwdclass'})}

**FAQ:** “If I override a field at the top, do I lose labels/error\_messages from Meta?”  
**No** — unless you explicitly set label or error\_messages on the override itself. Meta dictionaries still apply to overridden fields.

### Alternative (keep model-generated field, mutate it in \_\_init\_\_)

If you don’t want to “override” the field at class level:

class **RegistrationForm**(*forms***.***ModelForm*)**:**

    class **Meta:**

        model **=** Profile

        fields **=** ['name'**,** 'email'**,** 'passward']

        labels **=** {'name'**:** 'ENTER NAME'**,** 'email'**:** 'Enter Email:'**,** 'passward'**:** 'Enter Passward'}

        widgets **=** {'passward'**:** forms**.**PasswordInput(**attrs=**{'class'**:** 'pwdclass'})}

**def** \_\_init\_\_(*self***,** **\*args,** **\*\*kwargs**)**:**

        super()**.**\_\_init\_\_(**\***args**,** **\*\***kwargs)

*# tighten max\_length at form level*

*self***.**fields['name']**.**max\_length **=** 50

*# tweak attrs*

*self***.**fields['name']**.**widget**.**attrs**.**update({

            'class'**:** 'nameclass'**,**

            'placeholder'**:** 'enter your name '

        })

# 6) Adding an ****extra field not in the model**** (confirm password) + validate

class **RegistrationForm**(*forms***.***ModelForm*)**:**

*# extra (NOT in Profile model). It will NOT be saved by form.save()*

    conform\_passward **=** forms**.**CharField(  *# keeping your spelling, but usually: confirm\_password*

**widget=**forms**.**PasswordInput(**attrs=**{'placeholder'**:** 're-enter password'})

    )

    class **Meta:**

        model **=** Profile

        fields **=** ['name'**,** 'email'**,** 'passward']  *# model fields only; extra field is separate*

        labels **=** {'name'**:** 'ENTER NAME'**,** 'email'**:** 'Enter Email:'**,** 'passward'**:** 'Enter Passward'}

        widgets **=** {

            'passward'**:** forms**.**PasswordInput(**attrs=**{'class'**:** 'pwdclass'})**,**

            'name'**:** forms**.**TextInput(**attrs=**{'class'**:** 'nameclass'**,** 'placeholder'**:** 'enter your name '})

        }

*# Validate that passward == conform\_passward*

**def** clean(*self*)**:**

        cleaned **=** super()**.**clean()

        pwd **=** cleaned**.**get('passward')

        cpwd **=** cleaned**.**get('conform\_passward')

**if** pwd **and** cpwd **and** pwd **!=** cpwd**:**

*# attach error to the extra field (and/or the main field)*

*self***.**add\_error('conform\_passward'**,** 'Passwords do not match.')

**return** cleaned

The extra field is **only for validation/UI**. It won’t be saved by form.save() because it isn’t in Meta.fields. That’s exactly what we want.

# 7) views.py — Create, Update, Delete

### Create (three patterns)

*# student/views.py*

**from** django**.**shortcuts **import** render**,** redirect**,** get\_object\_or\_404

**from** student**.**forms **import** RegistrationForm

**from** student**.**models **import** Profile

**from** django**.**http **import** HttpResponseRedirect

**def** register(**request**)**:**

**if** request**.**method **==** 'POST'**:**

        form **=** RegistrationForm(request**.**POST)

**if** form**.**is\_valid()**:**

*# ✅ Preferred for ModelForm:*

*# If you need to mutate instance before saving (e.g., hash password), use commit=False*

            instance **=** form**.**save(**commit=**False)

*# ❗ SECURITY* **NOTE***:*

*# Never store clear-text passwords in real apps.*

*# Example of hashing (if you insisted on your own model):*

*# from django.contrib.auth.hashers import make\_password*

*# instance.passward = make\_password(instance.passward)*

            instance**.**save()  *# writes to DB*

*# If your form had M2M fields AND commit=False, call form.save\_m2m() after instance.save()*

*# form.save\_m2m()*

**return** redirect('/student/register/')  *# PRG pattern*

**else:**

        form **=** RegistrationForm()

**return** render(request**,** 'student/register.html'**,** {'form'**:** form})

**Important correction:** The commit=False flag belongs to **form.save(commit=False)**, not to model’s .save().  
Calling Profile(...).save(commit=False) is **wrong**—Model.save() does not accept commit.

### Update – Method 1 (manual get + assign)

**def** update\_profile\_manual(**request,** **pk**)**:**

    obj **=** get\_object\_or\_404(Profile**,** **pk=**pk)

**if** request**.**method **==** 'POST'**:**

*# Manual approach (without ModelForm)*

        obj**.**name **=** request**.**POST**.**get('name'**,** obj**.**name)

        obj**.**email **=** request**.**POST**.**get('email'**,** obj**.**email)

        obj**.**passward **=** request**.**POST**.**get('passward'**,** obj**.**passward)

        obj**.**save()

**return** redirect('/student/register/')

**return** render(request**,** 'student/update\_manual.html'**,** {'obj'**:** obj})

### Update – Method 2 (⚡ best with ModelForm and instance=)

**def** update\_profile(**request,** **pk**)**:**

    obj **=** get\_object\_or\_404(Profile**,** **pk=**pk)

**if** request**.**method **==** 'POST'**:**

        form **=** RegistrationForm(request**.**POST**,** **instance=**obj)  *# bind to existing row*

**if** form**.**is\_valid()**:**

            form**.**save()  *# updates that row*

**return** redirect('/student/register/')

**else:**

        form **=** RegistrationForm(**instance=**obj)  *# prefill with existing data*

**return** render(request**,** 'student/register.html'**,** {'form'**:** form})

Why “method 2” is better: You get all the field/constraint validation from both **form** and **model**, and you don’t risk forgetting a field. Also, if you do Profile(id=1, ...) and call .save(), Django will UPDATE that row **if it exists**; but if it **doesn’t** exist and your DB allows setting PK manually, you might accidentally **insert** with that id. Using instance= avoids that hazard.

### Delete

**def** delete\_profile(**request,** **pk**)**:**

    obj **=** get\_object\_or\_404(Profile**,** **pk=**pk)

**if** request**.**method **==** 'POST'**:**

        obj**.**delete()

**return** redirect('/student/register/')

**return** render(request**,** 'student/confirm\_delete.html'**,** {'obj'**:** obj})

# 8) Template (register.html)

Minimal version (renders all fields, including your extra conform\_passward):

**<**!-- templates**/**student**/**register**.**html --**>**

**<**!DOCTYPE html**>**

**<**html**>**

**<**head**>**

**<**meta charset**=**"utf-8"**>**

**<**title**>**Registration**</**title**>**

**<**style**>**

**.**errorlist { color**:** crimson; }  **/\*** default UL**-**based error *list* **\*/**

**.**field { margin**-**bottom**:** 12px; }

    label { display**:**block; font**-**weight**:**600; margin**-**bottom**:**4px; }

**</**style**>**

**</**head**>**

**<**body**>**

**<**h1**>**Register**</**h1**>**

**<**form method**=**"post" novalidate**>**

    {**%** csrf\_token **%**}

**<**!-- Top**-**level (non**-**field) errors**,** **if** any --**>**

    {{ form**.**non\_field\_errors }}

**<**!-- Render fields one by one **for** full control --**>**

**<**div **class**="field"**>**

      {{ form**.**name**.**label\_tag }} {{ form**.**name }}

      {{ form**.**name**.**errors }}

**</**div**>**

**<**div **class**="field"**>**

      {{ form**.**email**.**label\_tag }} {{ form**.**email }}

      {{ form**.**email**.**errors }}

**</**div**>**

**<**div **class**="field"**>**

      {{ form**.**passward**.**label\_tag }} {{ form**.**passward }}

      {{ form**.**passward**.**errors }}

**</**div**>**

**<**!-- Extra field **not** **in** model --**>**

**<**div class**=**"field"**>**

**<**label **for**="{{ form.conform\_passward.id\_for\_label }}"**>**Confirm Passward**</**label**>**

      {{ form**.**conform\_passward }}

      {{ form**.**conform\_passward**.**errors }}

**</**div**>**

**<**button *type*="submit"**>**Save**</**button**>**

**</**form**>**

**</**body**>**

**</**html**>**

# 9) fields='\_\_all\_\_' and exclude

* fields = '\_\_all\_\_' → include **every editable** model field.
* exclude = ['some\_field'] → include everything **except** the listed fields.
* **Never** set both fields and exclude in the same form.

Example:

class **RegistrationForm**(*forms***.***ModelForm*)**:**

    conform\_passward **=** forms**.**CharField(**widget=**forms**.**PasswordInput)

    class **Meta:**

        model **=** Profile

        fields **=** '\_\_all\_\_'          *# includes: name, email, passward*

*# exclude = ['passward']    # (don’t use together with fields)*

# 10) Redirect after POST (the PRG pattern)

You already did this; it’s the right call. After a successful POST, use redirect(...) or HttpResponseRedirect(...) to avoid the **“Confirm form resubmission”** browser prompt when refreshing.

# 11) A few important professional tips

* **Validation order** in ModelForm:
  1. Field clean (per field)
  2. Form.clean() (cross-field checks like password confirmation)
  3. Model validation (unique constraints, unique\_together (now UniqueConstraint), etc.)
* If you use **ManyToManyField** and call form.save(commit=False), remember to call form.save\_m2m() **after** saving the instance.
* Overriding a field on the form **narrows** the validation (e.g., max\_length=50), but does not change the DB schema (255). If you really want to change the DB, change the model and run migrations.
* Use \*\*Meta.help\_texts = {'field': '...'}\*\* for inline help in forms.
* To **reorder** fields, either order them in Meta.fields or pass field\_order=['...'] when instantiating the form.
* **Security:** Never store real passwords in plain text. Use Django’s built-in User model (or a custom user model) and its authentication system.

# 12) Full “everything in one place” version (with your exact ideas)

*# student/forms.py*

**from** django **import** forms

**from** student**.**models **import** Profile

class **RegistrationForm**(*forms***.***ModelForm*)**:**

*# extra field not in model*

    conform\_passward **=** forms**.**CharField(**widget=**forms**.**PasswordInput)

*# override a model field to tighten validation and add attrs*

    name **=** forms**.**CharField(

**max\_length=**50**,**

**widget=**forms**.**TextInput(**attrs=**{'class'**:** 'nameclass'**,** 'placeholder'**:** 'enter your name '})

    )

    class **Meta:**

        model **=** Profile

*# you can also use '\_\_all\_\_', then exclude via 'exclude' (but not both together)*

        fields **=** ['name'**,** 'email'**,** 'passward']

        labels **=** {'name'**:** 'ENTER NAME'**,** 'email'**:** 'Enter Email:'**,** 'passward'**:** 'Enter Passward'}

        error\_messages **=** {

            'email'**:** {'required'**:** 'Email is required'}

        }

        widgets **=** {

            'passward'**:** forms**.**PasswordInput(**attrs=**{'class'**:** 'pwdclass'})**,**

*# name widget also set above via field override; Meta.widgets would apply if not overridden*

        }

**def** clean(*self*)**:**

        cleaned **=** super()**.**clean()

        pwd **=** cleaned**.**get('passward')

        cpwd **=** cleaned**.**get('conform\_passward')

**if** pwd **and** cpwd **and** pwd **!=** cpwd**:**

*self***.**add\_error('conform\_passward'**,** 'Passwords do not match.')

**return** cleaned

*# student/views.py*

**from** django**.**shortcuts **import** render**,** redirect**,** get\_object\_or\_404

**from** student**.**forms **import** RegistrationForm

**from** student**.**models **import** Profile

**def** register(**request**)**:**

**if** request**.**method **==** 'POST'**:**

        form **=** RegistrationForm(request**.**POST)

**if** form**.**is\_valid()**:**

*# Typical ModelForm save*

            instance **=** form**.**save(**commit=**False)

*# Example place to transform passward (hashing, etc.)*

*# from django.contrib.auth.hashers import make\_password*

*# instance.passward = make\_password(instance.passward)*

            instance**.**save()

*# form.save\_m2m()  # only needed if there are M2M fields + commit=False*

**return** redirect('/student/register/')

**else:**

        form **=** RegistrationForm()

**return** render(request**,** 'student/register.html'**,** {'form'**:** form})

**def** update\_profile(**request,** **pk**)**:**

    obj **=** get\_object\_or\_404(Profile**,** **pk=**pk)

**if** request**.**method **==** 'POST'**:**

        form **=** RegistrationForm(request**.**POST**,** **instance=**obj)

**if** form**.**is\_valid()**:**

            form**.**save()

**return** redirect('/student/register/')

**else:**

        form **=** RegistrationForm(**instance=**obj)

**return** render(request**,** 'student/register.html'**,** {'form'**:** form})

**def** delete\_profile(**request,** **pk**)**:**

    obj **=** get\_object\_or\_404(Profile**,** **pk=**pk)

**if** request**.**method **==** 'POST'**:**

        obj**.**delete()

**return** redirect('/student/register/')

**return** render(request**,** 'student/confirm\_delete.html'**,** {'obj'**:** obj})

**<**!-- templates**/**student**/**register**.**html --**>**

**<**!DOCTYPE html**>**

**<**html**>**

**<**head**>**

**<**meta charset**=**"utf-8"**>**

**<**title**>**Registration**</**title**>**

**<**style**>**

**.**errorlist { color**:** crimson; }

**.**field { margin**-**bottom**:** 12px; }

    label { display**:**block; font**-**weight**:**600; margin**-**bottom**:**4px; }

**</**style**>**

**</**head**>**

**<**body**>**

**<**h1**>**Register (ModelForm)**</**h1**>**

**<**form method**=**"post" novalidate**>**

    {**%** csrf\_token **%**}

    {{ form**.**non\_field\_errors }}

**<**div class**=**"field"**>**

      {{ form**.**name**.**label\_tag }} {{ form**.**name }}

      {{ form**.**name**.**errors }}

**</**div**>**

**<**div class**=**"field"**>**

      {{ form**.**email**.**label\_tag }} {{ form**.**email }}

      {{ form**.**email**.**errors }}

**</**div**>**

**<**div class**=**"field"**>**

      {{ form**.**passward**.**label\_tag }} {{ form**.**passward }}

      {{ form**.**passward**.**errors }}

**</**div**>**

**<**div class**=**"field"**>**

**<**label **for**="{{ form.conform\_passward.id\_for\_label }}"**>**Confirm Passward**</**label**>**

      {{ form**.**conform\_passward }}

      {{ form**.**conform\_passward**.**errors }}

**</**div**>**

**<**button *type*="submit"**>**Save**</**button**>**

**</**form**>**

**</**body**>**

**</**html**>**

## Quick “why/when” recap

* Use **ModelForm** whenever your form corresponds to a database model row you want to **create or update**. You get:
  + automatic field generation,
  + automatic model validation (including unique checks),
  + simple form.save() to write to DB,
  + instance= to update safely.
* Use **form.save(commit=False)** when you need to **modify the instance before saving** (e.g., hash passwords, set created\_by, etc.). Then call .save() (and .save\_m2m() if needed).
* Overriding a field on the form **narrows/changes the form’s behavior** but **doesn’t change the DB column**. That’s OK and common.
* Extra fields (like conform\_passward) live only in the form; they’re great for **cross-field validation** inside clean().