CRUD application in Django:

- **Create** → `record_create`
- **Read/Display** → `record_display`
- **Update** → `record_update`
- **Delete** → `record_delete`

Here is a full **step-by-step guide** to build the project from scratch.

Create project:

```
django-admin startproject crudproject
```

Go into project folder:

```
cd crudproject
```

Create app:

```
python manage.py startapp records
```

Step 2: Register App

Open:

```
crudproject/settings.py
```

Add `'records'` inside `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'records',
]
```

Include App in project

crudproject/urls.py code:-

```
from django.contrib import admin
from django.urls import path,include



urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('records.urls')),
]
```

Step 3: Create Model

records/models.py file code:-

```
from django.db import models

class Record(models.Model):
    name = models.CharField(max_length=100)
    city = models.CharField(max_length=100)

    def __str__(self):
        return self.name
```

Step 4: Run Migrations

Run:

cd crudproject

```
python manage.py makemigrations
python manage.py migrate
```

records/forms.py file code:-

```
from django import forms
from .models import Record

class RecordForm(forms.ModelForm):
    class Meta:
        model=Record
        fields=['name','city']
```

records/views.py file code:-

```python
from django.shortcuts import render, redirect, get_object_or_404
from .models import Record
from .forms import RecordForm


# CREATE
def record_create(request):

    if request.method == 'POST':
        form = RecordForm(request.POST)

        if form.is_valid():
            form.save()
            return redirect('record_display')

    else:
        form = RecordForm()

    return render(request, 'records/record_create.html', {
        'form': form
    })


# READ / DISPLAY
def record_display(request):

    records = Record.objects.all()

    return render(request, 'records/record_display.html', {
        'records': records
    })
```

```python
# UPDATE
def record_update(request, pk):

    record = get_object_or_404(Record, pk=pk)

    if request.method == 'POST':
        form = RecordForm(request.POST, instance=record)

        if form.is_valid():
            form.save()
            return redirect('record_display')

    else:
        form = RecordForm(instance=record)

    return render(request, 'records/record_update.html', {
        'form': form
    })


# DELETE
def record_delete(request, pk):

    record = get_object_or_404(Record, pk=pk)

    record.delete()

    return redirect('record_display')
```

records/urls.py file code :-

```
from django.urls import path
from . import views

urlpatterns = [

    path('create/', views.record_create, name='record_create'),

    path('display/', views.record_display, name='record_display'),

    path('update/<int:pk>/', views.record_update, name='record_update'),

    path('delete/<int:pk>/', views.record_delete, name='record_delete'),

]
```

Templates files create templates folder inside your records app and create records folder inside templates folder as shown below:-

records/templates/records/

record_display.html file code:-

```
<h2>Records Table</h2>

<a href="{% url 'record_create' %}">
    Add New Record
</a>

<br><br>

<table border="1" cellpadding="10">

    <tr>
        <th>ID</th>
        <th>Name</th>
        <th>City</th>
        <th>Actions</th>
    </tr>

    {% for record in records %}

    <tr>
```

```
        <td>{{ record.id }}</td>

        <td>{{ record.name }}</td>

        <td>{{ record.city }}</td>

        <td>

            <a href="{% url 'record_update' record.id %}">
                Edit
            </a>

            |

            <a href="{% url 'record_delete' record.id %}">
                Delete
            </a>

        </td>



    </tr>

    {% endfor %}

</table>
```

create_record.html file:-

```
<h2>Add Record</h2>

<form method="POST">

    {% csrf_token %}

    {{ form.as_p }}

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

</form>

<br>

<a href="{% url 'record_display' %}">
    View Records
```

```
</a>
```

update_record.html file:-

```
<h2>Update Record</h2>

<form method="POST">

    {% csrf_token %}

    {{ form.as_p }}

    <button type="submit">
        Update
    </button>

</form>

<br>

<a href="{% url 'record_display' %}">
    Back to Records
</a>
```

Meaning of each line of code of views.py file code:-

Here is the **meaning of each line** of your Django CRUD code explained simply.

Import Statements

```
from django.shortcuts import render, redirect, get_object_or_404
```

Meaning:

This imports useful Django shortcut functions.

Function	Purpose
`render()`	Displays an HTML page
`redirect()`	Redirects user to another URL
`get_object_or_404()`	Gets object from database or shows 404 error

```
from .models import Record
```

Meaning:

Imports the `Record` model from `models.py`.

This lets you work with the database table.

Example:

```
Record.objects.all()
```

gets all records from database.

```
from .forms import RecordForm
```

Meaning:

Imports the Django form you created.

Used to:

- take user input
- validate data

- save data

CREATE FUNCTION

```
def record_create(request):
```

Meaning:

Creates a function called `record_create`.

This function handles creating new records.

`request` contains all browser request information.

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

Meaning:

Checks if user submitted the form.

POST means:

- user clicked submit
- data sent to server

```
form = RecordForm(request.POST)
```

Meaning:

Creates form object using submitted data.

`request.POST` contains form data.

Example:

```
name = John
email = john@gmail.com
age = 22
```

```
if form.is_valid():
```

Meaning:

Checks if form data is correct.

Validation checks:

- required fields
- email format
- valid age etc.

```
form.save()
```

Meaning:

Saves form data into database.

New record gets created.

```
return redirect('record_display')
```

Meaning:

After saving, redirect user to display page.

Instead of staying on same form page.

```
else:
```

Meaning:

Runs when page opens first time (GET request).

Not submitted yet.

```
form = RecordForm()
```

Meaning:

Creates empty form.

Used to display blank form to user.

```
return render(request, 'records/record_create.html', {
    'form': form
})
```

Meaning:

Displays HTML page.

Sends form object to template.

Template can access:

```
{{ form }}
```

READ / DISPLAY FUNCTION

```
def record_display(request):
```

Meaning:

Function to show all records.

```
records = Record.objects.all()
```

Meaning:

Gets all records from database.

Example:

Name	Email
John	john@gmail.com
Alex	alex@gmail.com

```
return render(request, 'records/record_display.html', {
    'records': records
})
```

Meaning:

Displays HTML page with all records.

Passes records to template.

In HTML:

```
{% for record in records %}
```

loops through them.

UPDATE FUNCTION

```
def record_update(request, pk):
```

Meaning:

Function to update existing record.

`pk` = Primary Key (ID).

Example:

```
/update/1/
```

Here:

```
pk = 1
```

```
record = get_object_or_404(Record, pk=pk)
```

Meaning:

Gets record with matching ID.

Example:

```
Record.objects.get(id=1)
```

If record not found:

Shows 404 error page.

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

Meaning:

Checks if updated form submitted.

```
form = RecordForm(request.POST, instance=record)
```

Meaning:

Loads submitted data into existing record.

`instance=record` means:

"update this record instead of creating new one"

```
if form.is_valid():
```

Meaning:

Checks updated data validity.

```
form.save()
```

Meaning:

Saves updated changes to database.

```
return redirect('record_display')
```

Meaning:

After update, go back to records list.

```
else:
```

Meaning:

When update page first opens.

```
form = RecordForm(instance=record)
```

Meaning:

Loads existing record data into form.

Form fields become pre-filled.

Example:

```
Name: John
Email: john@gmail.com
```

```
return render(request, 'records/record_update.html', {
    'form': form
})
```

Meaning:

Displays update form page.

DELETE FUNCTION

```
def record_delete(request, pk):
```

Meaning:

Function to delete a record.

```
record = get_object_or_404(Record, pk=pk)
```

Meaning:

Gets record by ID.

```
record.delete()
```

Meaning:

Deletes record from database.

```
return redirect('record_display')
```

Meaning:

After deletion, redirect to records list.

Overall Flow of Your CRUD App

```
CREATE  -> Add new data
READ    -> Show all data
UPDATE  -> Edit existing data
DELETE  -> Remove data
```

Simple Visual Understanding

```
User opens form
       ↓
User enters data
       ↓
POST request sent
       ↓
Form validation
       ↓
Save into database
       ↓
Redirect to display page
```

Main Django Concepts Used

Concept	Meaning
View Function	Handles request
Model	Database table
Form	User input handling
Template	HTML page
URL	Route to view
Redirect	Move to another page
Render	Show HTML

Example Real-Life Analogy

Think of:

Django Part	Real-Life Example
Model	Excel sheet
Form	Application form
View	Office worker
Template	Printed page
Database	Storage room

Final Summary

Your code performs complete CRUD operations:

Function	Job
`record_create`	Add new record
`record_display`	Show records
`record_update`	Edit record
`record_delete`	Delete record

Here is the **meaning of each line of your Django `urls.py` code**, explained simply and clearly.

□ Full Code Explanation Line by Line

```
from django.urls import path
```

Meaning:

- Imports Django's `path` function
- `path()` is used to create URL routes
- It connects a URL to a view (function)

```
from . import views
```

Meaning:

- Imports `views.py` from the current app
- `.` means same folder (your app)
- So you can access functions like:
  - `views.record_create`
  - `views.record_display`

□ URL Patterns List

```
urlpatterns = [
```

Meaning:

- This is a list of all URLs in your app
- Django checks this list to match user requests

Example:

```
/create/ → record_create view
```

□ CREATE URL

```
path('create/', views.record_create, name='record_create'),
```

Meaning:

- `'create/'` → URL path in browser

    - Example: `http://127.0.0.1:8000/create/`
- `views.record_create` → function that runs
- `name='record_create'` → shortcut name for this URL

Why name is used?

So you can write in HTML:

```
{% url 'record_create' %}
```

instead of full URL.

□ DISPLAY URL

```
path('display/', views.record_display, name='record_display'),
```

Meaning:

- URL: `/display/`
- Calls function: `record_display`
- Name: `record_display`

What it does:

Shows all records from database

□ UPDATE URL

```
path('update/<int:pk>/', views.record_update, name='record_update'),
```

Meaning:

This is a dynamic URL.

Breakdown:

- `'update/'` → base URL
- `<int:pk>` → dynamic value (integer ID)
- `pk` = primary key (record ID)

Example:
`/update/5/`

means:

- update record with ID = 5

Connected view:
`views.record_update`

- runs update function

Name:
`record_update`

- used in templates:

```
{% url 'record_update' record.id %}
```

□ DELETE URL

```
path('delete/<int:pk>/', views.record_delete, name='record_delete'),
```

Meaning:

- URL: `/delete/<id>/`

Breakdown:

- `'delete/'` → base path
- `<int:pk>` → record ID
- `views.record_delete` → function that deletes record
- `name='record_delete'` → shortcut name

Example:
`/delete/3/`

means:

- delete record with ID = 3

□ FULL SIMPLE SUMMARY

Line	Meaning
`path()`	creates URL
`create/`	add new record page
`display/`	show all records
`update/<int:pk>/`	edit specific record
`delete/<int:pk>/`	remove specific record
`name=`	shortcut for URL

□ SIMPLE FLOW

```
User enters URL
      ↓
Django checks urlpatterns
      ↓
Matches path
      ↓
Runs corresponding view
      ↓
Returns HTML page
```

□ REAL LIFE ANALOGY

Think of it like:

Django URL	Real Life
`/create/`	Add new student form
`/display/`	Student list page
`/update/5/`	Edit student #5
`/delete/5/`	Remove student #5

Here is the **meaning of each line of your Django HTML template code** explained simply and clearly.

□ Full Code Line-by-Line Explanation

```
<form method="POST">
```

Meaning:

- Starts an HTML form
- `method="POST"` means:
  - send data securely to server
  - used when saving or updating data

```
{% csrf_token %}
```

Meaning:

- Django security feature
- Prevents **CSRF attack (Cross Site Request Forgery)**
- Required for every POST form in Django

□ Without this, Django will give an error

```
{{ form.as_p }}
```

Meaning:

- Displays Django form fields automatically
- `as_p` means:
  - each field is wrapped in `<p>` tag

Example output:

```
<p>Name input</p>
<p>Email input</p>
<p>Age input</p>
```

□ Instead of writing HTML manually, Django generates it

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

Meaning:

- Creates a button
- `type="submit"` means:
  - sends form data to server when clicked
- Button text = “Save”

```
</form>
```

Meaning:

- Closes the form tag
- Marks end of input section

```
<br>
```

Meaning:

- Adds a line break (space)
- Moves next content to new line

```
<a href="{% url 'record_display' %}">
```

Meaning:

- Creates a clickable link
- `{% url 'record_display' %}` means:
  - Django will generate URL for named route `record_display`
  - usually `/display/`

 This avoids hardcoding URLs

```
View Records
```

Meaning:

- Text shown on the link
- User will see “View Records”

```
</a>
```

Meaning:

- Closes the hyperlink tag

□ SIMPLE FLOW OF THIS PAGE

```
User opens form page
      ↓
Django shows form fields
      ↓
User enters data
      ↓
Clicks Save button
      ↓
Data sent using POST
      ↓
Saved in database
      ↓
User can click "View Records"
```

□ KEY CONCEPTS USED

Code	Meaning
`<form>`	Input form container
POST	Sends data securely
csrf_token	Security protection
form.as_p	Auto-generate form fields
button submit	Sends data
url tag	Dynamic URL generation

□ REAL LIFE ANALOGY

Think of it like:

HTML Code	Real Life
Form	Application form
CSRF token	Security stamp
Submit button	“Submit application” button
View Records link	“Go to list page” sign