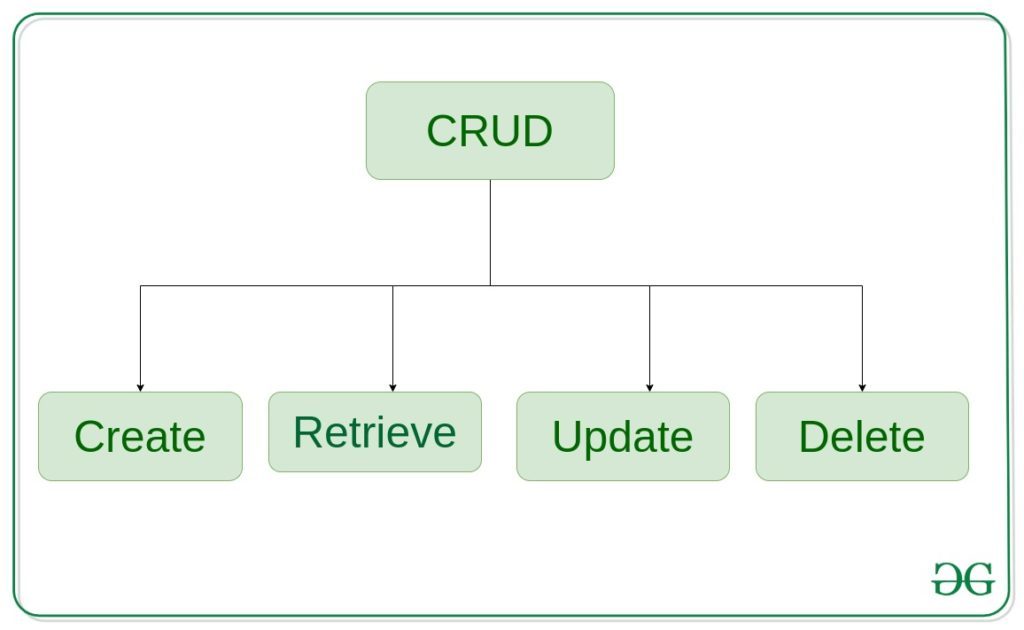
Django is a Python-based web framework which allows you to quickly create web application without all of the installation or dependency problems that you normally will find with other frameworks. Django is based on MVT (Model View Template) architecture and revolves around CRUD (Create, Retrieve, Update, Delete) operations. CRUD can be best explained as an approach to building a Django web application. In general CRUD means performing Create, Retrieve, Update and Delete operations on a table in a database. Let’s discuss what actually CRUD means,



**Create** – create or add new entries in a table in the database.   
**Retrieve** – read, retrieve, search, or view existing entries as a list(List View) or retrieve a particular entry in detail (Detail View)   
**Update** – update or edit existing entries in a table in the database   
**Delete** – delete, deactivate, or remove existing entries in a table in the database

**Django CRUD (Create, Retrieve, Update, Delete) Function Based Views**

Illustration of **How to create and use CRUD view** using an Example. Consider a project named geeksforgeeks having an app named geeks. 

*Refer to the following articles to check how to create a project and an app in Django.*

* [*How to Create a Basic Project using MVT in Django?*](about:blank)
* [*How to Create an App in Django ?*](about:blank)

After you have a project and an app, let’s create a model of which we will be creating instances through our view. In geeks/models.py, 

* Python3

|  |
| --- |
| # import the standard Django Model  # from built-in library  **from** django.db **import** models    # declare a new model with a name "GeeksModel"  **class** GeeksModel(models.Model):        # fields of the model      title **=** models.CharField(max\_length **=** 200)      description **=** models.TextField()        # renames the instances of the model      # with their title name  **def** \_\_str\_\_(self):  **return** self.title |

After creating this model, we need to run two commands in order to create Database for the same. 

Python manage.py [makemigrations](about:blank)

Python manage.py [migrate](about:blank)

Now we will create a Django ModelForm for this model. Refer this article for more on modelform – [Django ModelForm – Create form from Models](about:blank). create a file forms.py in geeks folder, 

* Python3

|  |
| --- |
| **from** django **import** forms  **from** .models **import** GeeksModel      # creating a form  **class** GeeksForm(forms.ModelForm):        # create meta class  **class** Meta:          # specify model to be used          model **=** GeeksModel            # specify fields to be used          fields **=** [              "title",              "description",          ] |

**Create View**

Create View refers to a view (logic) to create an instance of a table in the database. It is just like taking an input from a user and storing it in a specified table.   
In geeks/views.py, 

* Python3

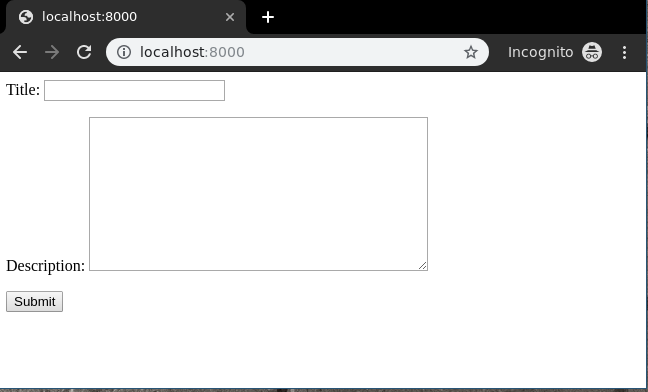
|  |
| --- |
| **from** django.shortcuts **import** render    # relative import of forms  **from** .models **import** GeeksModel  **from** .forms **import** GeeksForm      **def** create\_view(request):      # dictionary for initial data with      # field names as keys      context **=**{}        # add the dictionary during initialization      form **=** GeeksForm(request.POST **or** None)  **if** form.is\_valid():          form.save()        context['form']**=** form  **return** render(request, "create\_view.html", context) |

Create a template in templates/create\_view.html, 

* html

|  |
| --- |
| <**form** method="POST" enctype="multipart/form-data">        <!-- Security token -->      {% csrf\_token %}        <!-- Using the formset -->      {{ form.as\_p }}        <**input** type="submit" value="Submit">  </**form**> |

Now visit [http://localhost:8000/](about:blank) 



To check complete implementation of Function based Create View, visit [Create View – Function based Views Django](about:blank).

**Retrieve View**

Retrieve view is basically divided into two types of views Detail View and List View. 

**List View**

List View refers to a view (logic) to list all or particular instances of a table from the database in a particular order. It is used to display multiple types of data on a single page or view, for example, products on an eCommerce page.   
In geeks/views.py, 

* Python3

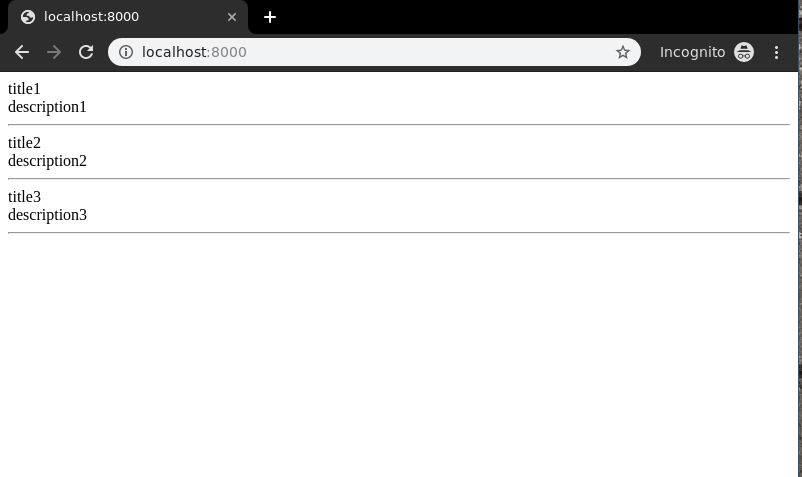
|  |
| --- |
| **from** django.shortcuts **import** render    # relative import of forms  **from** .models **import** GeeksModel      **def** list\_view(request):      # dictionary for initial data with      # field names as keys      context **=**{}        # add the dictionary during initialization      context["dataset"] **=** GeeksModel.objects.all()    **return** render(request, "list\_view.html", context) |

Create a template in templates/list\_view.html, 

* html

|  |
| --- |
| <**div** class="main">        {% for data in dataset %}.        {{ data.title }}<**br**/>      {{ data.description }}<**br**/>      <**hr**/>        {% endfor %}    </**div**> |

Now visit [http://localhost:8000/](about:blank) 



To check complete implementation of Function based List View, visit [List View – Function based Views Django](about:blank) 

**Detail View**

Detail View refers to a view (logic) to display a particular instance of a table from the database with all the necessary details. It is used to display multiple types of data on a single page or view, for example, profile of a user.   
In geeks/views.py, 

* Python3

|  |
| --- |
| **from** django.urls **import** path    # importing views from views..py  **from** .views **import** detail\_view    urlpatterns **=** [      path('<id>', detail\_view ),  ] |

Let’s create a view and template for the same. In geeks/views.py,

* Python3

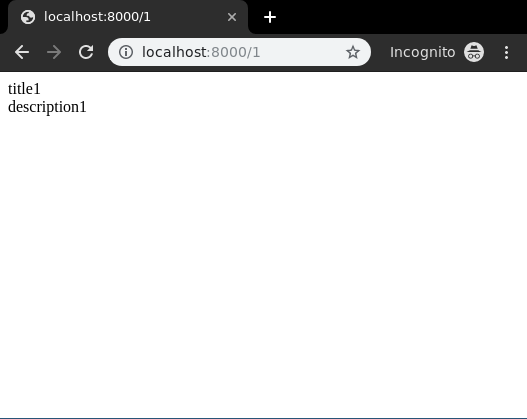
|  |
| --- |
| **from** django.shortcuts **import** render    # relative import of forms  **from** .models **import** GeeksModel    # pass id attribute from urls  **def** detail\_view(request, id):      # dictionary for initial data with      # field names as keys      context **=**{}        # add the dictionary during initialization      context["data"] **=** GeeksModel.objects.get(id **=** id)    **return** render(request, "detail\_view.html", context) |

Create a template in templates/Detail\_view.html, 

* html

|  |
| --- |
| <**div** class="main">        <!-- Specify fields to be displayed -->      {{ data.title }}<**br**/>      {{ data.description }}<**br**/>    </**div**> |

Let’s check what is there on [http://localhost:8000/1](about:blank) 



To check complete implementation of Function based Detail View, visit [Detail View – Function based Views Django](about:blank)

**Update View**

Update View refers to a view (logic) to update a particular instance of a table from the database with some extra details. It is used to update entries in the database for example, updating an article at geeksforgeeks.   
In geeks/views.py,

* Python3

|  |
| --- |
| **from** django.shortcuts **import** (get\_object\_or\_404,                                render,                                HttpResponseRedirect)    # relative import of forms  **from** .models **import** GeeksModel  **from** .forms **import** GeeksForm    # after updating it will redirect to detail\_View  **def** detail\_view(request, id):      # dictionary for initial data with      # field names as keys      context **=**{}        # add the dictionary during initialization      context["data"] **=** GeeksModel.objects.get(id **=** id)    **return** render(request, "detail\_view.html", context)    # update view for details  **def** update\_view(request, id):      # dictionary for initial data with      # field names as keys      context **=**{}        # fetch the object related to passed id      obj **=** get\_object\_or\_404(GeeksModel, id **=** id)        # pass the object as instance in form      form **=** GeeksForm(request.POST **or** None, instance **=** obj)        # save the data from the form and      # redirect to detail\_view  **if** form.is\_valid():          form.save()  **return** HttpResponseRedirect("/"**+**id)        # add form dictionary to context      context["form"] **=** form    **return** render(request, "update\_view.html", context) |

Now create following templates in templates folder,   
In geeks/templates/update\_view.html,

* html

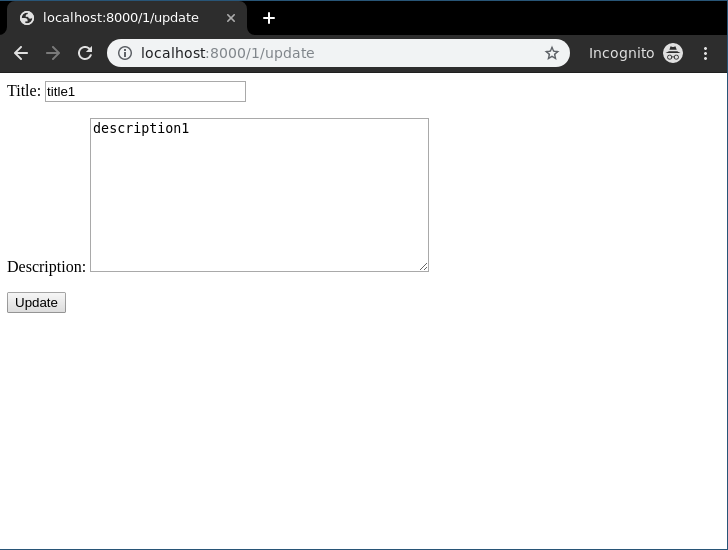
|  |
| --- |
| <**div** class="main">      <!-- Create a Form -->      <**form** method="POST">          <!-- Security token by Django -->          {% csrf\_token %}            <!-- form as paragraph -->          {{ form.as\_p }}            <**input** type="submit" value="Update">      </**form**>    </**div**> |

In geeks/templates/detail\_view.html, 

* html

|  |
| --- |
| <**div** class="main">      <!-- Display attributes of instance -->      {{ data.title }} <**br**/>      {{ data.description }}  </**div**> |

Let’s check if everything is working, visithttp://localhost:8000/1/update. 



To check complete implementation of Function based update View, visit Update View – Function based Views Django

**Delete View**

Delete View refers to a view (logic) to delete a particular instance of a table from the database. It is used to delete entries in the database for example, deleting an article at geeksforgeeks.   
In geeks/views.py 

* Python3

|  |
| --- |
| **from** django.shortcuts **import** (get\_object\_or\_404,                                render,                                HttpResponseRedirect)    **from** .models **import** GeeksModel      # delete view for details  **def** delete\_view(request, id):      # dictionary for initial data with      # field names as keys      context **=**{}        # fetch the object related to passed id      obj **=** get\_object\_or\_404(GeeksModel, id **=** id)      **if** request.method **==**"POST":          # delete object          obj.delete()          # after deleting redirect to          # home page  **return** HttpResponseRedirect("/")    **return** render(request, "delete\_view.html", context) |

Now a url mapping to this view with a regular expression of id,   
In geeks/urls.py 

* Python3

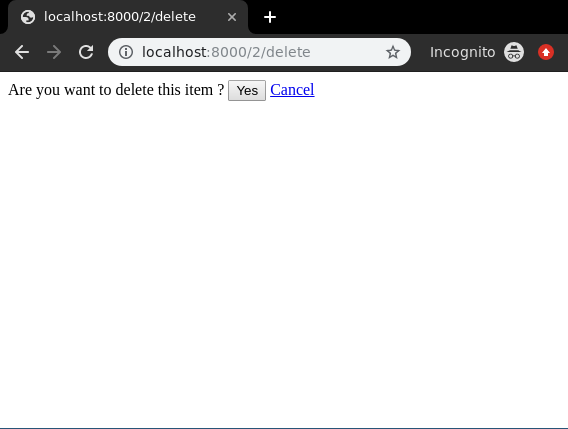
|  |
| --- |
| **from** django.urls **import** path    # importing views from views..py  **from** .views **import** delete\_view  urlpatterns **=** [      path('<id>/delete', delete\_view ),  ] |

Template for delete view includes a simple form confirming whether user wants to delete the instance or not. In geeks/templates/delete\_view.html, 

* html

|  |
| --- |
| <**div** class="main">      <!-- Create a Form -->      <**form** method="POST">          <!-- Security token by Django -->          {% csrf\_token %}          Are you want to delete this item ?          <**input** type="submit" value="Yes" />          <**a** href="/">Cancel </**a**>      </**form**>  </**div**> |

Everything ready, now let’s check if it is working or not, visit [http://localhost:8000/2/delete](about:blank) 



To check complete implementation of Function based Delete View, visit [Delete View – Function based Views Django](about:blank)

# Delete View – Function based Views Django

* **Difficulty Level :** Medium
* **Last Updated :** 17 May, 2021

 Read

 Discuss

 Courses

 Practice

 Video

Delete View refers to a view (logic) to delete a particular instance of a table from the database. It is used to delete entries in the database for example, deleting an article at geeksforgeeks. So Delete view must show a confirmation message to the user and should delete the instance automatically. Django provides extra-ordinary support for Delete Views but let’s check how it is done manually through a function-based view. This article revolves around Delete View which involves concepts such as [Django Forms](about:blank), [Django Models](about:blank).

For Delete View, we need a project with some models and multiple instances that we can use for deleting.

## Django Delete View – Function Based Views

Illustration of **How to create and use Delete view** using an Example. Consider a project named geeksforgeeks having an app named geeks.

*Refer to the following articles to check how to create a project and an app in Django.*

* [*How to Create a Basic Project using MVT in Django?*](about:blank)
* [*How to Create an App in Django ?*](about:blank)

After you have a project and an app, let’s create a model of which we will be creating instances through our view. In geeks/models.py,

* Python3

|  |
| --- |
| # import the standard Django Model  # from built-in library  **from** django.db **import** models    # declare a new model with a name "GeeksModel"  **class** GeeksModel(models.Model):        # fields of the model      title **=** models.CharField(max\_length **=** 200)      description **=** models.TextField()        # renames the instances of the model      # with their title name  **def** \_\_str\_\_(self):  **return** self.title |

After creating this model, we need to run two commands in order to create Database for the same.

Python manage.py [makemigrations](about:blank)

Python manage.py [migrate](about:blank)

Now let’s create some instances of this model using shell, run form bash,

Python manage.py shell

Enter following commands

>>> from geeks.models import GeeksModel

>>> GeeksModel.objects.create(

title="title1",

description="description1").save()

>>> GeeksModel.objects.create(

title="title2",

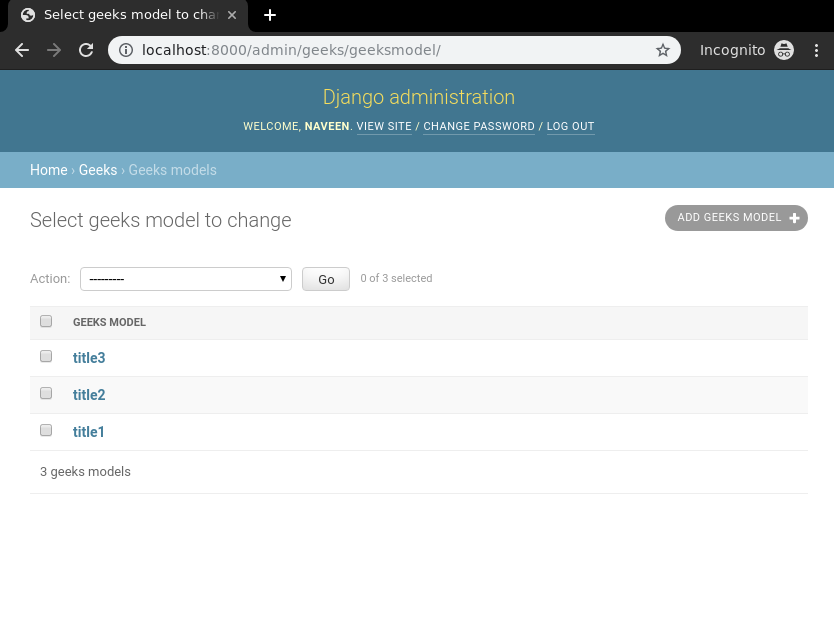
description="description2").save()

>>> GeeksModel.objects.create(

title="title2",

description="description2").save()

Now we have everything ready for back end. Verify that instances have been created from [http://localhost:8000/admin/geeks/geeksmodel/](about:blank)



Now let’s create our delete view first, In geeks/views.py

* Python3

|  |
| --- |
| **from** django.shortcuts **import** (get\_object\_or\_404,                                render,                                HttpResponseRedirect)    **from** .models **import** GeeksModel      # delete view for details  **def** delete\_view(request, id):      # dictionary for initial data with      # field names as keys      context **=**{}        # fetch the object related to passed id      obj **=** get\_object\_or\_404(GeeksModel, id **=** id)      **if** request.method **==**"POST":          # delete object          obj.delete()          # after deleting redirect to          # home page  **return** HttpResponseRedirect("/")    **return** render(request, "delete\_view.html", context) |

Now a url mapping to this view with a regular expression of id,   
In geeks/urls.py

* Python3

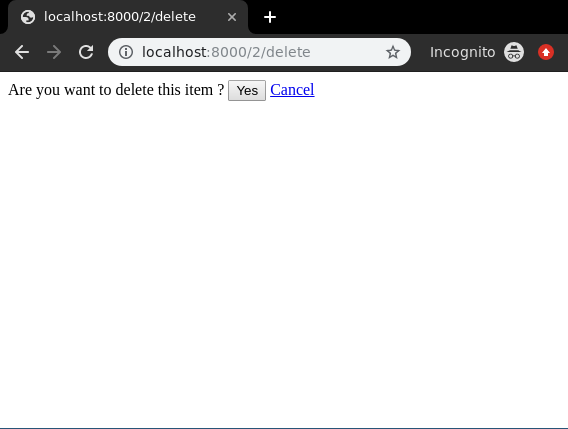
|  |
| --- |
| **from** django.urls **import** path    # importing views from views..py  **from** .views **import** delete\_view  urlpatterns **=** [      path('<id>/delete', delete\_view ),  ] |

Template for delete view includes a simple form confirming whether user wants to delete the instance or not. In geeks/templates/delete\_view.html,

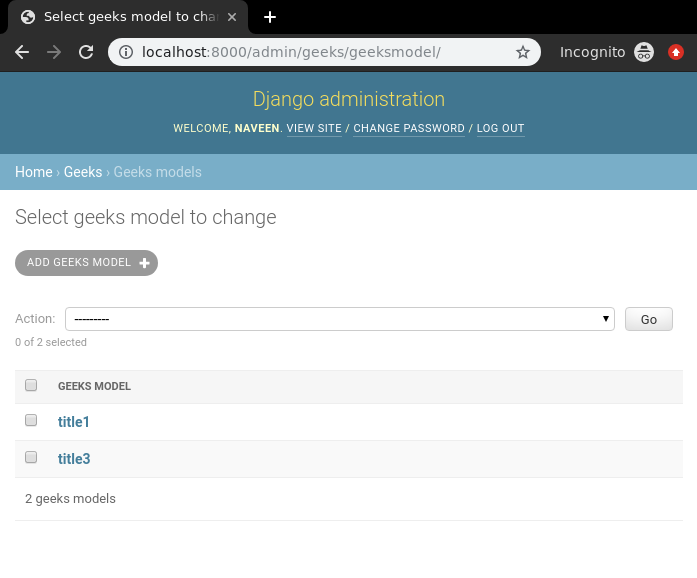
* HTML

|  |
| --- |
| <**div** class="main">      <!-- Create a Form -->      <**form** method="POST">          <!-- Security token by Django -->          {% csrf\_token %}          Are you want to delete this item ?          <**input** type="submit" value="Yes" />          <**a** href="/">Cancel </**a**>      </**form**>  </**div**> |

Everything ready, now let’s check if it is working or not, visit [http://localhost:8000/2/delete](about:blank)



Let’s check if instance has been deleted or not,



One can implement this view in any manner as per requirement using **obj.delete()** function.