Chapter-10

Class Based Views And CRUD Operations By Using Both CBVs And FBVs

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Class Based Views (CBVs):

There are 2 types of views

1) **Function Based Views**
2) **Class Based Views**

Note:

1) **Class Based Views introduced in Django 1.3 Version to implement Generic Views.**
2) **When compared with Function Based views,class Based views are very easy to use.Hence Class Based Views are Most frequently used views in real time.**
3) **Interanally Class Based Views will be converted into Function Based Views.Hence Class Based Views are simply acts as wrappers to the Function based views to hide complexity.**
4) **Function Based views are more powerful when compared with Class Based Views.**

Q) Explain the Sceneario where we should use Function based Views only and we cannot use Class based Views?

- **For simple operations like listing of all records or display details of a particular record then we should go for Class Based Views.**
- **For complex operations like handling multiple forms simulataneously then we should go for Function Based Views.**

  Eg: KFC

HelloWorld Application By using ClassBasedViews

views.py

```
1)  from django.views.generic import View
2)  from django.http import HttpResponse
3)
4)  # Create your views here.
5)  class HelloWorldView(View):
6)      def get(self,request):
7)          return HttpResponse('<h1>This is from ClassBasedView</h1>')
```

urls.py

```
1)  from testapp import views
2)
3)  urlpatterns = [
4)      ...
5)      url(r'^$', views.HelloWorldView.as_view()),
6)  ]
```

Note:

1) While defining Class Based Views we have to extend View class.
2) To provide response to GET request, Django will always call get() method.Hence we have to override this method to provide response to the GET request.Similarly for other HTTP Methods also like POST,HEAD etc
3) While defining url pattern we have to use as_view() method.

Template Based Application by using Class Based Views:

```
class TemplateCBV(TemplateView)
   template_name = 'home.html'
```

How to send Context Paramters:

```
class TemplateCBV(TemplateView)
   template_name = 'home.html'

def get_context_data(self,**kwargs)
      context = super().get_context_data(**kwargs)
      context['name'] = 'durga'
      context['age'] = 30
      return context
```

In template file we can access context parameters as follows

```
{{name}}
{{age}}
```

ListView:

We can use ListView class to list out all records from database table(model).

It is alternative to `ModelClassName.objects.all()`

models.py

```
1)  from django.db import models
2)
3)  # Create your models here.
4)  class Book(models.Model):
5)      title=models.CharField(max_length=300)
6)      author=models.CharField(max_length=30)
7)      pages=models.IntegerField()
8)      price=models.FloatField()
```

views.py

```
1)  from testapp.models import Book
2)  from django.views.generic import ListView
3)
4)  # Create your views here.
5)  class BookListView(ListView):
6)      model=Book
```

How to create Template File for ListView:

Django will identify template automatically and we are not required to configure anywhere. But Django will always search for template file with the name modelclassname_list.html like book_list.html

Django will always search for template file in the following location.
projectname/appname/templates/appname/
Eg: cbvproject5/testapp/templates/test/book_list.html

Note: bydefault django will provide context object to the template file with the name: modelclassname_list
Eg: book_list

book_list.html

```
1)  <!DOCTYPE html>
2)  <html lang="en" dir="ltr">
3)    <head>
4)      <meta charset="utf-8">
5)      <title></title>
6)    </head>
7)    <body>
8)      <h1>All Books Information</h1><hr>
9)      {% for book in book_list%}
10)     <ul>
11)       <li> <strong>Title</strong>:{{book.title}}</li>
12)       <li> <strong>Author</strong>:{{book.author}}</li>
13)       <li> <strong>Pages</strong>:{{book.pages}}</li>
14)       <li> <strong>Price</strong>:{{book.price}}</li>
15)     </ul>
16)     <hr>
17)     {%endfor%}
18)   </body>
19) </html>
```

How to provide our own Context Object Name:

The default context object name is: modelclassname_list
But we can customize this name based on our requirement as follows

```
class BookListView(ListView):
    context_object_name = 'books'
    model = Book
```

How to configure our own Template File at Project Level:

ofcourse this approach is not recommended

```
class BookListView(ListView):
    context_object_name = 'books'
    model = Book
    template_name = 'testapp/durga.html'
```

Note: Even if we are not specifying template_name variable,still django can recognize project level template file. But name should be modelclassname_list.html

DetailView:

We can use ListView to list of all records present in the database table.
But to get details of a particular record, we should go for DetailView.

models.py

```
1)  from django.db import models
2)
3)  class Company(models.Model):
4)      name=models.CharField(max_length=128)
5)      location=models.CharField(max_length=64)
6)      ceo=models.CharField(max_length=64)
```

admin.py

```
1)  from django.contrib import admin
2)  from testapp.models import Company
3)
4)  # Register your models here.
5)  class CompanyAdmin(admin.ModelAdmin):
6)      list_display=['name','location','ceo']
7)
8)  admin.site.register(Company,CompanyAdmin)
```

views.py

```
1)  from django.shortcuts import render
2)  from testapp.models import Company
3)  from django.views.generic import ListView,DetailView
4)
5)  # Create your views here.
6)  class CompanyListView(ListView):
7)      model=Company
8)      #default template_name is company_list.html
9)      #defult context_object_name is company_list
10)
11) class CompanyDetailView(DetailView):
12)     model=Company
13)     #default template_name is company_detail.html
14)     #defult context_object_name is company or object
```

base.html

```
1)  <!DOCTYPE html>
2)  <html lang="en" dir="ltr">
3)    <head>
4)      <meta charset="utf-8">
5)      <title></title>
6)      <!-- Latest compiled and minified CSS -->
7)  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/cs
    s/bootstrap.min.css" integrity="sha384-
    BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" c
    rossorigin="anonymous">
8)    </head>
9)    <body>
10)     <div class="container" >
11)       {%block body_block%}
12)       {%endblock %}
13)     </div>
14)   </body>
15) </html>
```

company_list.html

```
1)  <!DOCTYPE html>
2)  {%extends 'testapp/base.html'%}
3)        {%block body_block%}
4)          <h1>List of All Companies</h1><hr>
5)          <ol>
```

```
6)          {%for company in company_list%}
7)        <h2><li> <a href="/{{company.id}}">{{company.name|upper}}</a> </li></h2>
8)          {%endfor%}
9)         </ol>
10)      {%endblock %}
```

company_detail.html

```
1)  <!DOCTYPE html>
2)  {%extends 'testapp/base.html'%}
3)       {%block body_block%}
4)         <h1>Company Information</h1><hr>
5)         <ol>
6)           <h2><li>Company Name: {{company.name|upper}}</li></h2>
7)           <h2><li>Company Location: {{company.location|title}}</li></h2>
8)           <h2><li>Company CEO: {{company.ceo|title}}</li></h2>
9)         </ol>
10)      {%endblock %}
```

urls.py

```
1)  from django.conf.urls import url
2)  from django.contrib import admin
3)  from testapp import views
4)
5)  urlpatterns = [
6)     url(r'^admin/', admin.site.urls),
7)     url(r'^companies/', views.CompanyListView.as_view()),
8)     url(r'^(?P<pk>\d+)/$', views.CompanyDetailView.as_view()),
9)  ]
```

Adding Employee Information also

models.py

```
1)  from django.db import models
2)
3)  # Create your models here.
4)  class Company(models.Model):
5)     name=models.CharField(max_length=128)
6)     location=models.CharField(max_length=64)
7)     ceo=models.CharField(max_length=64)
8)
9)     def __str__(self):
```

```
10)       return self.name
11)
12) class Employee(models.Model):
13)    eno=models.IntegerField()
14)    name=models.CharField(max_length=128)
15)    salary=models.FloatField()
16)    company=models.ForeignKey(Company,related_name='employees')
```

company_detail.html

```
1)  <!DOCTYPE html>
2)  {%extends 'testapp/base.html'%}
3)        {%block body_block%}
4)          <h1>Company Information</h1><hr>
5)          <ol>
6)            <h2><li>Company Name: {{company.name|upper}}</li></h2>
7)            <h2><li>Company Location: {{company.location|title}}</li></h2>
8)            <h2><li>Company CEO: {{company.ceo|title}}</li></h2>
9)            <h2>Employee Information</h2>
10)           {%for emp in company.employees.all%}
11)            <ul>
12)              <li>Employee Number:{{emp.eno}}</li>
13)              <li>Employee Name:{{emp.name}}</li>
14)              <li>Employee Salary:{{emp.salary}}</li> <hr>
15)            </ul>
16)           {%endfor%}
17)         </ol>
18)  {%endblock %}
```

Django CRUD Operations

1) C → Create (Insert Query)
2) R → Retrieve (Select Query)
3) U → Update (Update Query)
4) D → Delete (Delete Query)

For any web application,it is very common requirement to perform CRUD operations.

Case Study: BookMyshow Application

1) Add New Movie Information (Create)
2) Show Movie Information (Retrieve)
3) Update New timings for existing Movie (Update)
4) Delete old Movie Information (Delete)

By using the following ClassBased Views we can perform CRUD operations very easily.

ListView, DetailView → Retrieve Operation
CreateView → Create Operation (Insert Data)
UpdateView → Update Operation
DeleteView → Delete Operation

CreateView Class:

We can use this CreateView class to insert data into our models.

views.py

```
1) class CompanyCreateView(CreateView):
2)     model=Company
```

urls.py

```
1) urlpatterns = [
2)     url(r'^admin/', admin.site.urls),
3)     url(r'^companies/', views.CompanyListView.as_view()),
4)     url(r'^(?P<pk>\d+)/$', views.CompanyDetailView.as_view(),name='detail'),
5)     url(r'^create/', views.CompanyCreateView.as_view(),name='create'),
6) ]
```

If we send Request:

ImproperlyConfigured at /create/
Using ModelFormMixin (base class of CompanyCreateView) without the 'fields' attribute is prohibited.

We can solve this problem by defining fields attribute in CreateView class

```
class CompanyCreateView(CreateView):
    model = Company
    fields = ('name','location','ceo')
```

If we send Request:
TemplateDoesNotExist at /create/
testapp/company_form.html

By default CreateView class will always search for template file named with modelclassname_form.html
Eg: company_form.html

company_form.html

```
<!DOCTYPE html>
{%extends 'testapp/base.html'%}
    {%block body_block%}
     <h1>Company Create Form</h1><hr>
     <form  method="post">
          {{form.as_p}}
          {%csrf_token%}
       <input type="submit" class='btn btn-primary btn-lg' value="Insert Record">
     </form>
{%endblock %}
```

If we fill form and submit:

ImproperlyConfigured at /create/
No URL to redirect to. Either provide a URL or define a get_absolute_url method on the Model.

How to define get_absolute_url() in Model class:

```
from django.db import models
from django.core.urlresolvers import reverse

# Create your models here.
class Company(models.Model):
    name=models.CharField(max_length=128)
    location=models.CharField(max_length=64)
    ceo=models.CharField(max_length=64)
    def __str__(self):
        return self.name
def get_absolute_url(self):
        return reverse('detail',kwargs={'pk':self.pk})
```

UpdateView class:

We can use UpdateView to update existing record.

views.py

```
class CompanyUpdateView(UpdateView):
    model=Company
    fields=('name','ceo')
```

urls.py

Define URL for this updateview

http://localhost:8000/update/7

```
1)  from django.conf.urls import url
2)  from django.contrib import admin
3)  from testapp import views
4)
5)  urlpatterns = [
6)      url(r'^admin/', admin.site.urls),
7)      url(r'^companies/', views.CompanyListView.as_view()),
8)      url(r'^(?P<pk>\d+)/$', views.CompanyDetailView.as_view(),name='detail'),
9)      url(r'^create/', views.CompanyCreateView.as_view(),name='create'),
10)     url(r'^update/(?P<pk>\d+)/$', views.CompanyUpdateView.as_view(),name='upd
    ate'),
11) ]
```

Add Update Button in Details Page

company_detail.html

```
1)  <!DOCTYPE html>
2)  {%extends 'testapp/base.html'%}
3)      {%block body_block%}
4)        <h1>Company Information</h1><hr>
5)        <ol>
6)          <h2><li>Company Name: {{company.name|upper}}</li></h2>
7)          <h2><li>Company Location: {{company.location|title}}</li></h2>
8)          <h2><li>Company CEO: {{company.ceo|title}}</li></h2>
9)        </ol>
10)       <a href="/update/{{company.id}}" class='btn btn-warning'>Update</a>
11)     {%endblock %}
```

DeleteView class:

We can use DeleteView to delete records

views.py

```
1)  from django.core.urlresolvers import reverse_lazy
2)  class CompanyDeleteView(DeleteView):
3)      model=Company
4)      success_url=reverse_lazy('/companies')
```

success_url represents the target page which should be displayed after delete.
reverse_lazy() function will wait until deleting the record.

urls.py

```
url(r'^delete/(?P<pk>\d+)/$', views.CompanyDeleteView.as_view(),name='delete')
```

http://localhost:8000/delete/7

TemplateDoesNotExist at /delete/7/
testapp/company_confirm_delete.html

If we are trying to delete, then DeleteView will provide confirmation template.
The default template name is model_confirm_delete.html
Eg: company_confirm_delete.html
We have to provide this template file.

company_confirm_delete.html

```
1)  <!DOCTYPE html>
2)  {%extends 'testapp/base.html'%}
3)      {%block body_block%}
4)       <h1>Delete {{company.name}} ???</h1><hr>
5)       <form method="post">
6)        {%csrf_token%}
7)        <input type="submit" class='btn btn-danger' value="Delete Record">
8)        <a href="/{{company.id}}" class='btn btn-success'>Cancel</a>
9)       </form>
10)     {%endblock %}
```

To Place Delete Button on Details Page

```
1)  <!DOCTYPE html>
2)  {%extends 'testapp/base.html'%}
3)      {%block body_block%}
4)       <h1>Company Information</h1><hr>
5)       <ol>
6)        <h2><li>Company Name: {{company.name|upper}}</li></h2>
7)        <h2><li>Company Location: {{company.location|title}}</li></h2>
8)        <h2><li>Company CEO: {{company.ceo|title}}</li></h2>
9)       </ol>
10)      <a href="/update/{{company.id}}" class='btn btn-warning'>Update</a>
11)      <a href="/delete/{{company.id}}" class='btn btn-danger'>Delete</a>
12)
13)     {%endblock %}
```

Django Class Based Views Complete Example-1 (cbvproject7)

models.py

```
1)  from django.db import models
2)  from django.core.urlresolvers import reverse
3)
4)  # Create your models here.
5)  class Beer(models.Model):
6)      name=models.CharField(max_length=128)
7)      taste=models.CharField(max_length=128)
8)      color=models.CharField(max_length=128)
9)      price=models.FloatField()
10)     def __str__(self):
11)         return self.name
12)
13)     def get_absolute_url(self):
14)         return reverse('detail',kwargs={'pk':self.pk})
```

admin.py

```
1)  from django.contrib import admin
2)  from testapp.models import Company
3)
4)  # Register your models here.
5)  class CompanyAdmin(admin.ModelAdmin):
6)      list_display=['name','location','ceo']
7)  admin.site.register(Company,CompanyAdmin)
```

views.py

```
1)  from django.shortcuts import render
2)  from testapp.models import Company
3)  from django.core.urlresolvers import reverse_lazy
4)  from django.views.generic import ListView,DetailView,CreateView,UpdateView,
    DeleteView
5)
6)  # Create your views here.
7)  class CompanyListView(ListView):
8)      model=Company
9)      #default template_name is company_list.html
10)     #defult context_object_name is company_list
11) class CompanyDetailView(DetailView):
```

```
    model=Company
    #default template_name is company_detail.html
    #defult context_object_name is company or object
class CompanyCreateView(CreateView):
    model=Company
    fields=('name','location','ceo')

class CompanyUpdateView(UpdateView):
    model=Company
    fields=('name','ceo')

class CompanyDeleteView(DeleteView):
    model=Company
    success_url=reverse_lazy('companies')
```

urls.py

```
from django.conf.urls import url
from django.contrib import admin
from testapp import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^companies/', views.CompanyListView.as_view(),name='companies'),
    url(r'^(?P<pk>\d+)/$', views.CompanyDetailView.as_view(),name='detail'),
    url(r'^create/', views.CompanyCreateView.as_view(),name='create'),
    url(r'^update/(?P<pk>\d+)/$', views.CompanyUpdateView.as_view(),name=
'update'),
    url(r'^delete/(?P<pk>\d+)/$', views.CompanyDeleteView.as_view(),name='delete'),
]
```

base.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title></title>
    <!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/cs
s/bootstrap.min.css" integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
  </head>
  <body>
```

```
11)   <div class="container" >
12)    {%block body_block%}
13)    {%endblock %}
14)   </div>
15)  </body>
16) </html>
```

company_list.html

```
1)  <!DOCTYPE html>
2)  {%extends 'testapp/base.html'%}
3)        {%block body_block%}
4)          <h1>List of All Companies</h1><hr>
5)          <ol>
6)           {%for company in company_list%}
7)           <h2><li> <a href="/{{company.id}}">{{company.name|upper}}</a> </li></h2>
8)           {%endfor%}
9)          </ol>
10)       {%endblock %}
```

company_detail.html

```
1)  <!DOCTYPE html>
2)  {%extends 'testapp/base.html'%}
3)        {%block body_block%}
4)          <h1>Company Information</h1><hr>
5)          <ol>
6)           <h2><li>Company Name: {{company.name|upper}}</li></h2>
7)           <h2><li>Company Location: {{company.location|title}}</li></h2>
8)           <h2><li>Company CEO: {{company.ceo|title}}</li></h2>
9)          </ol>
10)         <a href="/update/{{company.id}}" class='btn btn-warning'>Update</a>
11)         <a href="/delete/{{company.id}}" class='btn btn-danger'>Delete</a>
12)       {%endblock %}
```

company_form.html

```
1)  <!DOCTYPE html>
2)  {%extends 'testapp/base.html'%}
3)        {%block body_block%}
4)          <h1>Company Create Form</h1><hr>
5)          <form  method="post">
6)           {{form.as_p}}
7)           {%csrf_token%}
8)           <input type="submit" class='btn btn-primary btn-lg' value="Insert Record">
```

```
        </form>
    {%endblock %}
```

company_confirm_delete.html

```
<!DOCTYPE html>
{%extends 'testapp/base.html'%}
    {%block body_block%}
      <h1>Delete {{company.name}} ???</h1><hr>
      <form method="post">
        {%csrf_token%}
        <input type="submit" class='btn btn-danger' value="Delete Record">
        <a href="/{{company.id}}" class='btn btn-success'>Cancel</a>
      </form>
    {%endblock %}
```

Django Class Based Views Complete Example (cbvfinalproject)

models.py

```
from django.db import models
from django.core.urlresolvers import reverse

# Create your models here.
class Beer(models.Model):
    name=models.CharField(max_length=128)
    taste=models.CharField(max_length=128)
    color=models.CharField(max_length=128)
    price=models.FloatField()

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('detail',kwargs={'pk':self.pk})
```

admin.py

```
from django.contrib import admin
from testapp.models import Beer

# Register your models here.
class BeerAdmin(admin.ModelAdmin):
    list_display=['name','taste','color','price']
```

```
7)  admin.site.register(Beer,BeerAdmin)
```

views.py

```
1)  from django.shortcuts import render
2)  from testapp.models import Beer
3)  from django.core.urlresolvers import reverse_lazy
4)  from django.views.generic import ListView,DetailView,CreateView,UpdateView,De
    leteView
5)
6)  # Create your views here.
7)  class BeerListView(ListView):
8)      model=Beer
9)
10) class BeerDetailView(DetailView):
11)     model=Beer
12)
13) class BeerCreateView(CreateView):
14)     model=Beer
15)     #fields=('name','taste','color','price')
16)     fields='__all__'
17) class BeerUpdateView(UpdateView):
18)     model=Beer
19)     fields=('taste','color','price')
20) class BeerDeleteView(DeleteView):
21)     model=Beer
22)     success_url=reverse_lazy('home')
```

urls.py

```
1)  from django.conf.urls import url
2)  from django.contrib import admin
3)  from testapp import views
4)
5)  urlpatterns = [
6)      url(r'^admin/', admin.site.urls),
7)      url(r'^$', views.BeerListView.as_view(),name='home'),
8)      url(r'^(?P<pk>\d+)/$', views.BeerDetailView.as_view(),name='detail'),
9)      url(r'^create/', views.BeerCreateView.as_view()),
10)     url(r'^update/(?P<pk>\d+)/$', views.BeerUpdateView.as_view()),
11)     url(r'^delete/(?P<pk>\d+)/$', views.BeerDeleteView.as_view()),
12) ]
```

base.html

```
1)  <!DOCTYPE html>
2)  <html lang="en" dir="ltr">
3)    <head>
4)      <meta charset="utf-8">
5)      <title></title>
6)      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7
    /css/bootstrap.min.css" integrity="sha384-
    BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" c
    rossorigin="anonymous">
7)    </head>
8)    <body>
9)      <div class="container">
10)       {%block body_block%}
11)       {%endblock%}
12)     </div>
13)   </body>
14) </html>
```

beer_list.html

```
1)  <!DOCTYPE html>
2)  {% extends 'testapp/base.html'%}
3)      {%block body_block%}
4)        <h1>Beer Information Dash Board</h1><hr>
5)        <table border='2'>
6)          <thead>
7)            <th>Beer Name</th>
8)            <th>Details</th>
9)            <th>Update</th>
10)           <th>Delete</th>
11)         </thead>
12)         {%for  beer in beer_list %}
13)         <tr>
14)           <td>{{beer.name|title}}</td>
15)           <td> <a href="/{{beer.id}}">Details</a> </td>
16)           <td><a href="/update/{{beer.id}}">Update</a></td>
17)           <td><a href="/delete/{{beer.id}}">Delete</a></td>
18)         </tr>
19)         {%endfor%}
20)       </table><br><br><br>
21)       <a href="/create" class='btn btn-primary btn-lg'>
    Do You Want to Insert New Beer</a>
22)     {%endblock%}
```

beer_detail.html

```
1)  <!DOCTYPE html>
2)  {% extends 'testapp/base.html'%}
3)      {%block body_block%}
4)        <div class="jumbotron">
5)          <h1>Beer Details...</h1><hr>
6)          <ul>
7)            <li>Beer Name: {{beer.name}}</li>
8)            <li>Beer Taste: {{beer.taste}}</li>
9)            <li>Beer Color: {{beer.color}}</li>
10)           <li>Beer Price: {{beer.price}}</li>
11)         </ul>
12)       </div>
13)     {%endblock%}
```

beer_form.html

```
1)  <!DOCTYPE html>
2)  {% extends 'testapp/base.html'%}
3)      {%block body_block%}
4)        <h1>Add New Beer Here</h1><hr>
5)        <form method="post">
6)          {{form.as_p}}
7)          {%csrf_token%}
8)          <input type="submit" class='btn btn-primary btn-lg'
    name="" value="Insert/Update">
9)        </form>
10)     {%endblock%}
```

beer_confirm_delte.html

```
1)  <!DOCTYPE html>
2)  {% extends 'testapp/base.html'%}
3)      {%block body_block%}
4)        <h1>Do You Want to Delete {{beer.name}} Record</h1><hr>
5)        <form method="post">
6)          {%csrf_token%}
7)          <input type="submit" class='btn btn-lg btn-danger' value="DELETE">
8)          <a href="/" class='btn btn-lg btn-success'>CANCEL</a>
9)        </form>
10)     {%endblock%}
```

CRUD Operations by using Function Based Views (FBVs)

1) Start Project
2) Start App
3) Templates → testapp → *.html
4) Add Application and Templates Path to settings.py
5) Create Employee Model Class
6) Make Migrations and Migrate
7) Register this Model inside admin.py and Create Super User
8) Execute populate Script

views.py

```
1)  from django.shortcuts import render,redirect
2)  from testapp.forms import EmployeeForm
3)  from testapp.models import Employee
4)
5)  # Create your views here.
6)  def show_view(request):
7)      employees=Employee.objects.all()
8)      return render(request,'testapp/index.html',{'employees':employees})
9)
10) def insert_view(request):
11)     form=EmployeeForm()
12)     if request.method=='POST':
13)         form=EmployeeForm(request.POST)
14)         if form.is_valid():
15)             form.save()
16)         return redirect('/')
17)     return render(request,'testapp/insert.html',{'form':form})
```

models.py

```
1)  from django.db import models
2)
3)  # Create your models here.
4)  class Employee(models.Model):
5)      eno=models.IntegerField()
6)      ename=models.CharField(max_length=64)
7)      esal=models.FloatField()
8)      eaddr=models.CharField(max_length=256)
```

forms.py

```
1)  from django import forms
2)  from testapp.models import Employee
3)  class EmployeeForm(forms.ModelForm):
4)      class Meta:
5)          model=Employee
6)          fields='__all__'
```

populate.py

```
1)  import os
2)  os.environ.setdefault('DJANGO_SETTINGS_MODULE','fbvproject1.settings')
3)  import django
4)  django.setup()
5)
6)  from testapp.models import *
7)  from faker import Faker
8)  from random import *
9)  faker=Faker()
10) def populate(n):
11)     for i in range(n):
12)         feno=randint(1001,9999)
13)         fename=faker.name()
14)         fesal=randint(10000,20000)
15)         feaddr=faker.city()
16)         emp_record=Employee.objects.get_or_create(eno=feno,ename=fename,esal=f
    esal,eaddr=feaddr)
17) populate(10)
```

views.py (Delete & Update)

```
1)  def delete_view(request,id):
2)      employee=Employee.objects.get(id=id)
3)      employee.delete()
4)      return redirect('/')
5)
6)  def update_view(request,id):
7)      employee=Employee.objects.get(id=id)
8)      if request.method=='POST':
9)          form=EmployeeForm(request.POST,instance=employee)
10)         if form.is_valid():
11)             form.save()
12)             return redirect('/')
13)     return render(request,'testapp/update.html',{'employee':employee})
```

Note: In the following line if we are not using instance then a new record will be created.

form = EmployeeForm(request.POST,instance = employee)

form = EmployeeForm(request.POST) → New Record will be created
form = EmployeeForm(request.POST, instance = employee) → Existing Record will be updated

Complete Application (fbvproject1)

models.py

```
1)  from django.db import models
2)  # Create your models here.
3)  class Employee(models.Model):
4)      eno=models.IntegerField()
5)      ename=models.CharField(max_length=64)
6)      esal=models.FloatField()
7)      eaddr=models.CharField(max_length=256)
```

admin.py

```
1)  from django.db import models
2)
3)  # Create your models here.
4)  class Employee(models.Model):
5)      eno=models.IntegerField()
6)      ename=models.CharField(max_length=64)
7)      esal=models.FloatField()
8)      eaddr=models.CharField(max_length=256)
```

views.py

```
1)  from django.shortcuts import render,redirect
2)  from testapp.forms import EmployeeForm
3)  from testapp.models import Employee
4)
5)  # Create your views here.
6)  def show_view(request):
7)      employees=Employee.objects.all()
8)      return render(request,'testapp/index.html',{'employees':employees})
9)
10) def insert_view(request):
11)     form=EmployeeForm()
```

```
12)    if request.method=='POST':
13)        form=EmployeeForm(request.POST)
14)        if form.is_valid():
15)            form.save()
16)        return redirect('/')
17)    return render(request,'testapp/insert.html',{'form':form})
18)
19) def delete_view(request,id):
20)    employee=Employee.objects.get(id=id)
21)    employee.delete()
22)    return redirect('/')
23)
24) def update_view(request,id):
25)    employee=Employee.objects.get(id=id)
26)    if request.method=='POST':
27)        form=EmployeeForm(request.POST,instance=employee)
28)        if form.is_valid():
29)            form.save()
30)            return redirect('/')
31)    return render(request,'testapp/update.html',{'employee':employee})
```

urls.py

```
1)  from django.conf.urls import url
2)  from django.contrib import admin
3)  from testapp import views
4)
5)  urlpatterns = [
6)      url(r'^admin/', admin.site.urls),
7)      url(r'^$', views.show_view),
8)      url(r'^insert/', views.insert_view),
9)      url(r'^delete/(?P<id>\d+)/$', views.delete_view),
10)     url(r'^update/(?P<id>\d+)/$', views.update_view),
11) ]
```

base.html

```
1)  <!DOCTYPE html>
2)  <html lang="en" dir="ltr">
3)    <head>
4)      <meta charset="utf-8">
5)      <title></title>
6)      <!-- Latest compiled and minified CSS -->
7)  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/
    css/bootstrap.min.css" integrity="sha384-
```

```
8)  BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" c
    rossorigin="anonymous">
9)   </head>
10)  <body>
11)   <div class="container" align='center'>
12)    {%block body_block%}
13)    {%endblock%}
14)   </div>
15)  </body>
16) </html>
```

index.html

```
1)  <!DOCTYPE html>
2)  {% extends 'testapp/base.html'%}
3)      {%block body_block%}
4)      <h1>Employee Information Dash Board</h1><hr>
5)      <table border='2'>
6)       <thead>
7)         <th>Employee Number</th>
8)         <th>Employee Name</th>
9)         <th>Employee Salary</th>
10)        <th>Employee Address</th>
11)        <th>Actions</th>
12)      </thead>
13)      {%for  emp in employees %}
14)      <tr>
15)        <td>{{emp.eno}}</td>
16)        <td>{{emp.ename}}</td>
17)        <td>{{emp.esal}}</td>
18)        <td>{{emp.eaddr}}</td>
19)        <td>
20)          <a href="/update/{{emp.id}}">Update</a>
21)          <a href="/delete/{{emp.id}}">Delete</a>
22)        </td>
23)      </tr>
24)      {%endfor%}
25)     </table><br><br><br>
26)     <a href="/insert" class='btn btn-primary btn-lg'>
    Do You Want to Insert New Employee</a>
27)     {%endblock%}
```

insert.html

```
1)  <!DOCTYPE html>
2)  {% extends 'testapp/base.html'%}
3)      {%block body_block%}
4)      <h1>Employee Insert Form</h1><hr>
5)      <form method="post">
6)       <table border='1'>
7)         {{form}}
8)       </table>
9)       {%csrf_token%}
10)      <br>
11)      <input type="submit" class='btn btn-success btn-lg' value="Insert Record">
12)     </form>
13)     {%endblock%}
```

update.html

```
1)  <!DOCTYPE html>
2)  {% extends 'testapp/base.html'%}
3)      {%block body_block%}
4)      <h1>Employee Update Form</h1><hr>
5)      <form  method="post">
6)       {%csrf_token%}
7)  Employee Number: <input type="text" name="eno" value="{{employee.eno}}">
    <p></p>
8)  Employee Name: <input type="text" name="ename" value="{{employee.ename}}"
    ><p></p>
9)  Employee Salary: <input type="text" name="esal" value="{{employee.esal}}">
    <p></p>
10) Employee Address: <input type="text" name="eaddr" value="{{employee.eaddr}}"
    > <p></p>
11) <input type="submit" class='btn btn-warning btn-lg' value="Update Record">
12)     </form>
13)     {%endblock%}
```

How to use django form for update:

In forms.py, create the form with instance employee as
"form = EmployeeForm(instance = employee) "
Send the form object intead of employee object in render function of update_views as
"{'form':form} "
 {{form.as_p}}

Differences between CBVs and FBVs

CBVs	FBVs
1) CBVs can be easily extended	1) FBVs cannot extended easily
2) CBVs promotes Reusability of the Code	2) FBVs cannot promote Reusability of the Code
3) CBVs can use Object Oriented Techniques such as Mixins (Multiple Inheritance)	3) FBVs cannot use Object Oriented Techniques
4) In CBVs, Less Coding	4) In FBVs, More Coding
5) Default Context Dictionary and Default Template Files Support Available	5) Default Context Dictionary and Default Template Files Support not Available
6) Handling HTTP Methods by seperate Class Methods such as get() and post()	6) Handling HTTP Methods via Conditional Braching if request.method == 'POST'
7) There is a Restriction on Functionality and hence Less Power.	7) Based on Requirement we can implement any Functionality and hence these are more Powerful
8) Implicit Execution Flow and hence reduces Readability.	8) Explicit Execution Flow and hence improves Readability.

Note: In Real Time the most commonly used views are CBVs.If CBV can not handle our requirement then only we should go for FBVs.