# UNIT IV : Basics of Django

## Django Model

In Django, a model is a class which is used to contain essential fields and methods. Each model class maps to a single table in the database.

Django Model is a subclass of **django.db.models.Model** and each field of the model class represents a database field (column).

Django provides us a database-abstraction API which allows us to create, retrieve, update and delete a record from the mapped table.

Model is defined in **Models.py** file. This file can contain multiple models.

Let's see an example here, we are creating a model **Employee** which has two fields **first_name** and **last_name**.

```
from django.db import models

class Employee(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

The **first_name** and **last_name** fields are specified as class attributes and each attribute maps to a database column.

This model will create a table into the database that looks like below.

```
CREATE TABLE appname_employee (
   "id" INT NOT NULL PRIMARY KEY,
   "first_name" varchar(30) NOT NULL,
   "last_name" varchar(30) NOT NULL
);
```

The created table contains an auto-created **id field**. The name of the table is a combination of app name and model name that can be changed further.

## Register / Use Model

After creating a model, register model into the **INSTALLED_APPS** inside **settings.py.**

**For example,**

```
INSTALLED_APPS = [
    #...
    'appname',
    #...
]
```

## Django Model Fields

The fields defined inside the Model class are the columns name of the mapped table.

| Field Name | Class | Particular |
|---|---|---|
| AutoField | class AutoField(**options) | It An IntegerField that automatically increments. |
| BooleanField | class BooleanField(**options) | A true/false field. The default form widget for this field is a CheckboxInput. |
| CharField | class DateField(auto_now=False, auto_now_add=False, **options) | It is a date, represented in Python by a datetime.date instance. |
| DateTimeField | class DateTimeField(auto_now=False, auto_now_add=False, **options) | It is a date, represented in Python by a datetime.date instance. |
| DecimalField | class DecimalField(max_digits=None, decimal_places=None, **options) | It is a fixed-precision decimal number, represented in Python by a Decimal instance. |
| FileField | class FileField(upload_to=None, max_length=100, **options) | It is a file-upload field. |
| FloatField | class FloatField(**options) | It is a floating-point number represented in Python by a float instance. |

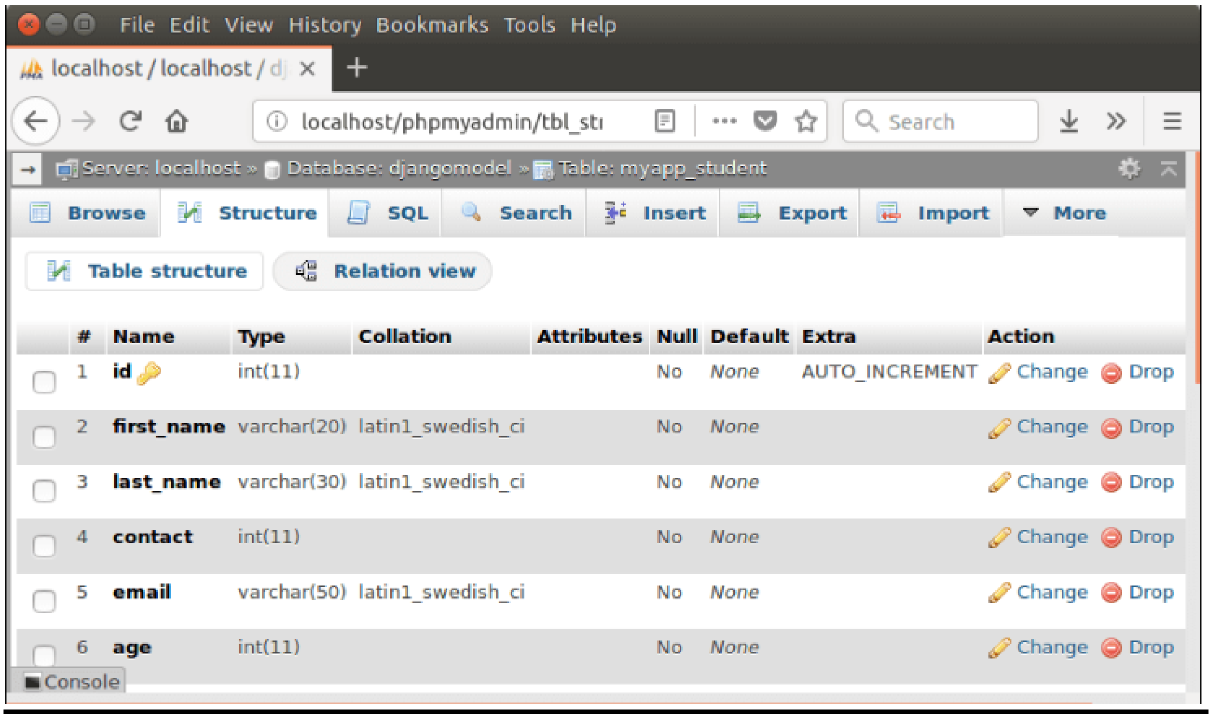
## Django Model Example

We created a model Student that contains the following code in **models.py** file.

**//models.py**

```
class Student(models.Model):
    first_name = models.CharField(max_length=20)
    last_name  = models.CharField(max_length=30)
    contact    = models.IntegerField()
    email      = models.EmailField(max_length=50)
    age        = models.IntegerField()
```

It will create a table **myapp_student**. The table structure looks like the below.



File Edit View History Bookmarks Tools Help
localhost / localhost / d
localhost/phpmyadmin/tbl_st
Search
Server: localhost » Database: djangomodel » Table: myapp_student
Browse
Structure
SQL
Search
Insert
Export
Import
More
Table structure
Relation view
Console

| # | Name | Type | Collation | Attributes | Null | Default | Extra | Action |
|---|---|---|---|---|---|---|---|---|
| 1 | id | int(11) | | | No | None | AUTO_INCREMENT | Change Drop |
| 2 | first_name | varchar(20) | latin1_swedish_ci | | No | None | | Change Drop |
| 3 | last_name | varchar(30) | latin1_swedish_ci | | No | None | | Change Drop |
| 4 | contact | int(11) | | | No | None | | Change Drop |
| 5 | email | varchar(50) | latin1_swedish_ci | | No | None | | Change Drop |
| 6 | age | int(11) | | | No | None | | Change Drop |

# Django Model Form

It is a class which is used to create an HTML form by using the Model. It is an efficient way to create a form without writing HTML code.

Django automatically does it for us to reduce the application development time. For example, suppose we have a model containing various fields, we don't need to repeat the fields in the form file.

For this reason, Django provides a helper class which allows us to create a Form class from a Django model.

## Django ModelForm Example

First, create a model that contains fields name and other metadata. It can be used to create a table in database and dynamic HTML form.

**// model.py**

```
from __future__ import unicode_literals
from django.db import models

class Student(models.Model):
    first_name = models.CharField(max_length=20)
    last_name  = models.CharField(max_length=30)
    class Meta:
        db_table = "student"    # Custom table name
```

This file contains a class that inherits ModelForm and mention the model name for which HTML form is created.

**// form.py**

```
from django import forms
from myapp.models import Student

class EmpForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = "__all__"
```

Write a view function to load the ModelForm from forms.py.

**//views.py**

```
from django.shortcuts import render
from myapp.form import StuForm

def index(request):
    stu = EmpForm()
    return render(request,"index.html",{'form':stu})
```

**//urls.py**

```
from django.contrib import admin
from django.urls import path
from myapp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', views.index),
1.  ]
2.
```

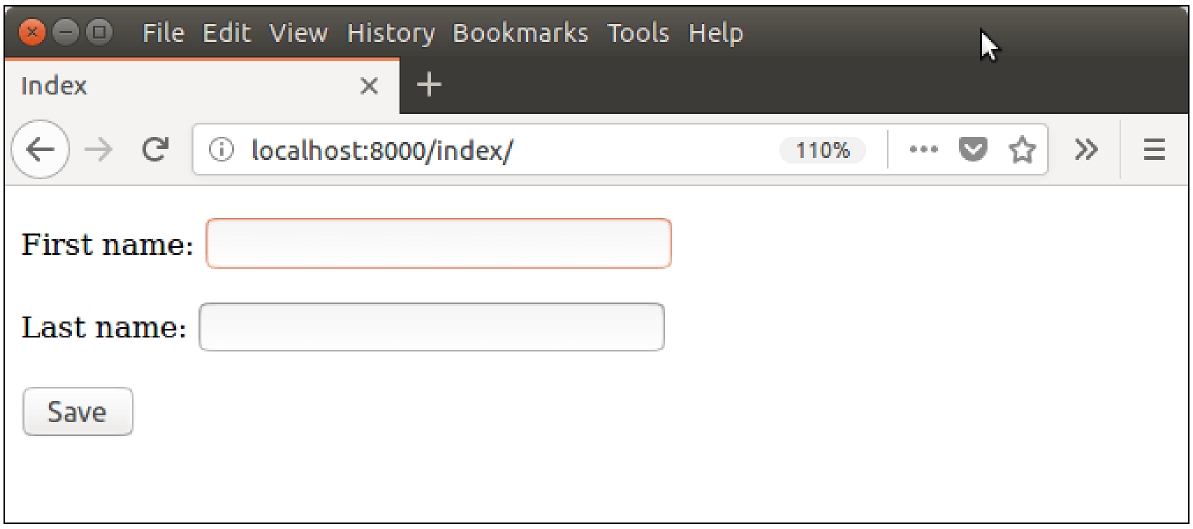
And finally, create a **index.html** file that contains the following code.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Index</title>
</head>
<body>
```

```
<form method="POST" class="post-form">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" >Save</button>
    </form>
</body>
</html>
```

## Run Server

Run the server by using **python manage.py runserver** command.



First name:

Last name:

Save

## Django Forms

Django provides a Form class which is used to create HTML forms. It describes a form and how it works and appears.

It is similar to the **ModelForm** class that creates a form by using the Model, but it does not require the Model.

Each field of the form class map to the HTML form **<input>** element and each one is a class itself, it manages form data and performs validation while submitting the form.

## Building a Form in Django

Suppose we want to create a form to get Student information, use the following code

**//forms.py**.

```
from django import forms
class StudentForm(forms.Form):
    firstname = forms.CharField(label="Enter first name",max_length=50)
    lastname  = forms.CharField(label="Enter last name", max_length = 100)
```

Note: Django Form does not include <form> tags, or a submit button. We'll have to provide those ourselves in the template.

## Instantiating Form in Django

Now, we need to instantiate the form in **views.py** file. See, the below code.

**// views.py**

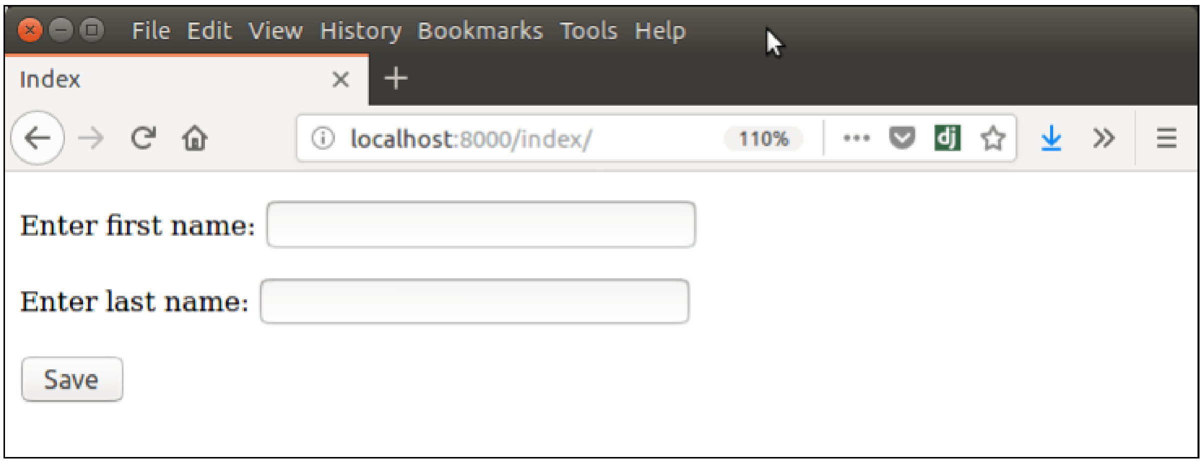
```
from django.shortcuts import render

from myapp.form import StudentForm


def index(request):

    student = StudentForm()

    return render(request,"index.html",{'form':student})
```

**// index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Index</title>
</head>
<body>
<form method="POST" class="post-form">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Save</button>
</form>
</body>
</html>
```

**//urls.py**

```
from django.contrib import admin
from django.urls import path
from myapp import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', views.index),
]
```



Enter first name:

Enter last name:

Save

## Django Form Validation

Django provides built-in methods to validate form data automatically. Django forms submit only if it contains CSRF tokens. It uses uses a clean and easy approach to validate data.

The **is_valid()** method is used to perform validation for each field of the form, it is defined in Django Form class. It returns True if data is valid and place all data into a cleaned_data attribute.

**// views.py**

```
from django.shortcuts import render

from myapp.form import StudentForm


def index(request):

 if request.method == "POST":

        form = StudentForm(request.POST)
        if form.is_valid():
            try:
                return redirect('/')
            except:
                pass
    else:
    return render(request,"index.html",{'form':student})
```

# Django Database Connectivity

The **settings.py** file contains all the project settings along with database connection details. By default, Django works with **SQLite,** database and allows configuring for other databases as well.

Database connectivity requires all the connection details such as database name, user credentials, hostname drive name etc

To connect with MySQL, **django.db.backends.mysql** driver is used to establishing a connection between application and database. Let's see an example.

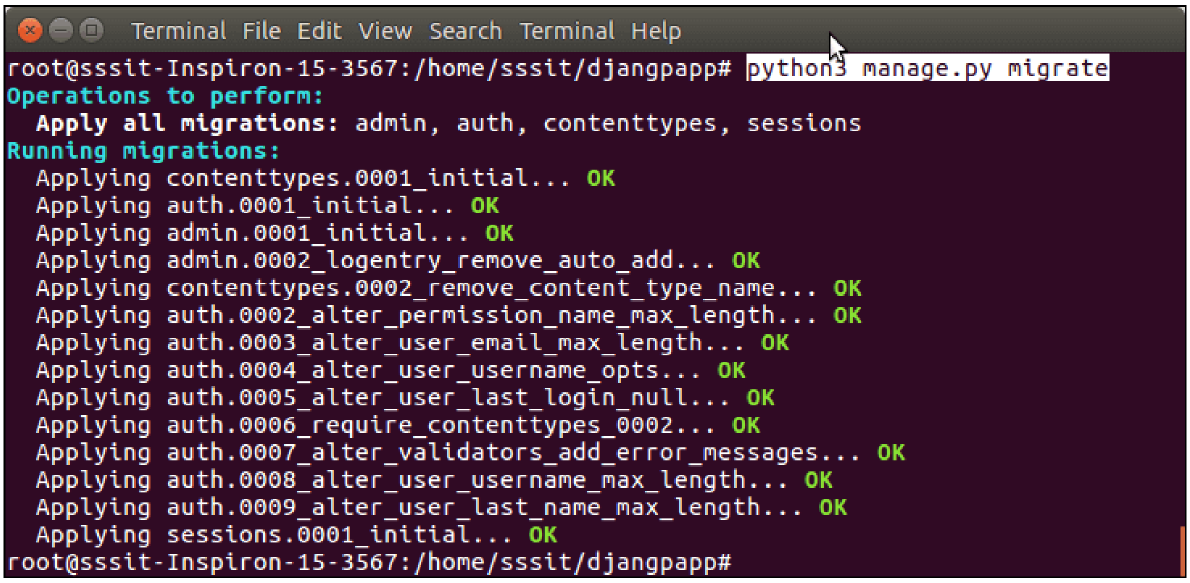
We need to provide all connection details in the settings file. The settings.py file of our project contains the following code for the database.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'djangoApp',
        'USER':'root',
        'PASSWORD':'mysql',
        'HOST':'localhost',
        'PORT':'3306'
    }
}
```

After providing details, check the connection using the migrate command.

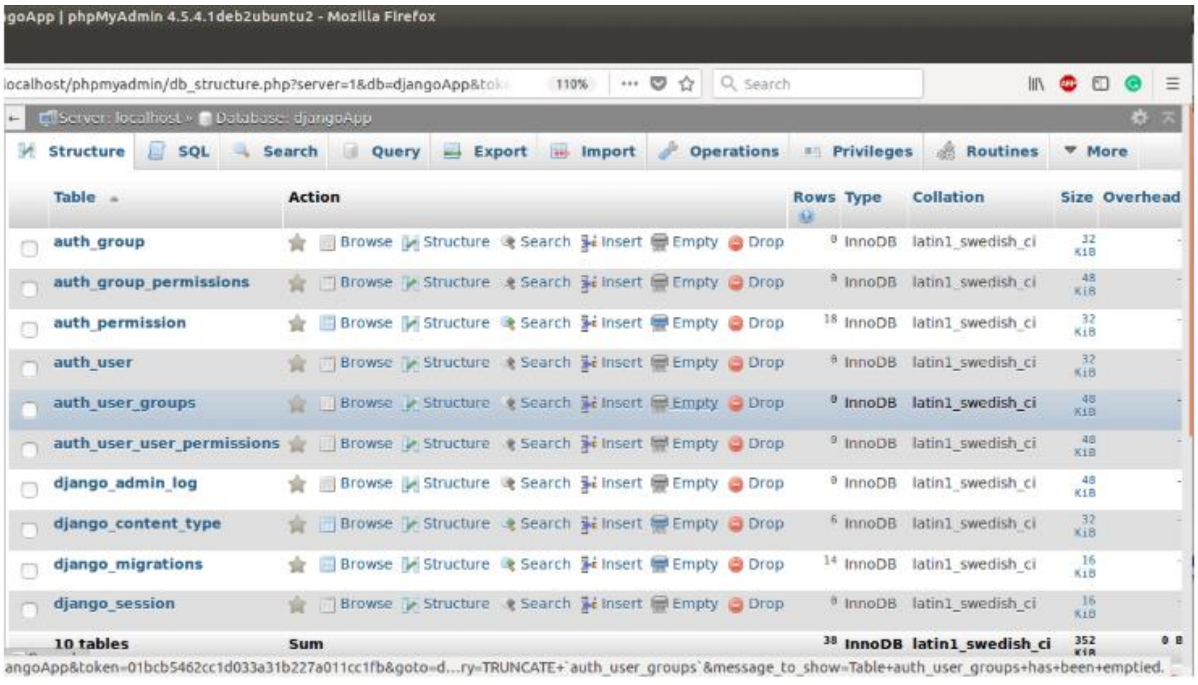
**$ python3 manage.py migrate**

This command will create tables for admin, auth, contenttypes, and sessions. See the example.



```
root@sssit-Inspiron-15-3567:/home/sssit/djangpapp# python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, 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 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 sessions.0001_initial... OK
root@sssit-Inspiron-15-3567:/home/sssit/djangpapp#
```

Now, access to the MySQL database and see the database from the list of databases. The created database contains the following tables.



goApp | phpMyAdmin 4.5.4.1deb2ubuntu2 - Mozilla Firefox

localhost/phpmyadmin/db_structure.php?server=1&db=djangoApp&tok

Server: localhost » Database: djangoApp

Structure | SQL | Search | Query | Export | Import | Operations | Privileges | Routines | More

| Table | Action | Rows | Type | Collation | Size | Overhead |
|---|---|---|---|---|---|---|
| auth_group | Browse Structure Search Insert Empty Drop | 0 | InnoDB | latin1_swedish_ci | 32 KiB | - |
| auth_group_permissions | Browse Structure Search Insert Empty Drop | 0 | InnoDB | latin1_swedish_ci | 48 KiB | - |
| auth_permission | Browse Structure Search Insert Empty Drop | 18 | InnoDB | latin1_swedish_ci | 32 KiB | - |
| auth_user | Browse Structure Search Insert Empty Drop | 0 | InnoDB | latin1_swedish_ci | 32 KiB | - |
| auth_user_groups | Browse Structure Search Insert Empty Drop | 0 | InnoDB | latin1_swedish_ci | 48 KiB | - |
| auth_user_user_permissions | Browse Structure Search Insert Empty Drop | 0 | InnoDB | latin1_swedish_ci | 48 KiB | - |
| django_admin_log | Browse Structure Search Insert Empty Drop | 0 | InnoDB | latin1_swedish_ci | 48 KiB | - |
| django_content_type | Browse Structure Search Insert Empty Drop | 6 | InnoDB | latin1_swedish_ci | 32 KiB | - |
| django_migrations | Browse Structure Search Insert Empty Drop | 14 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| django_session | Browse Structure Search Insert Empty Drop | 0 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| 10 tables | Sum | 38 | InnoDB | latin1_swedish_ci | 352 KiB | 0 B |

angoApp&token=01bcb5462cc1d033a31b227a011cc1fb&goto=d...ry=TRUNCATE+`auth_user_groups`&message_to_show=Table+auth_user_groups+has+been+emptied.

## Django CRUD (Create Read Update Delete) Application

Django provides an admin site to allow CRUD (Create Read Update Delete) operations on registered app model.

It is a built-in feature of Django that automatically generates interface for models.

We can see the url entry for admin in urls.py file, it is implicit and generated while creating a new project.

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

To create a Django application that performs CRUD operations, follow the following steps.

**1. Create a Project**

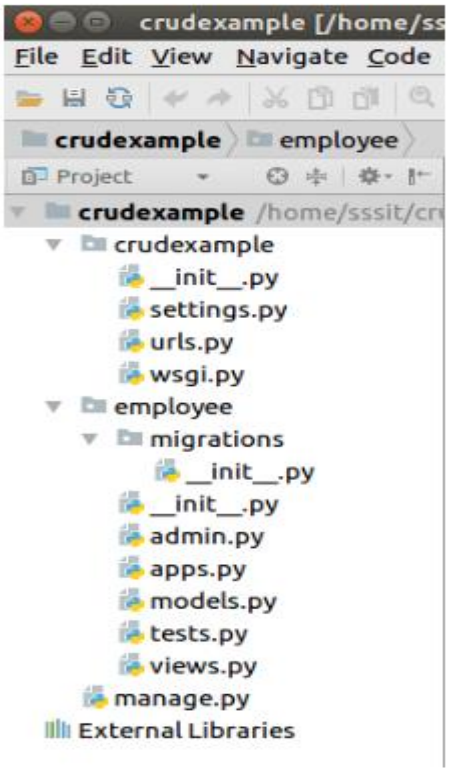
```
$ django-admin startproject crudexample
```

**2. Create an App**

```
$ python3 manage.py startapp employee
```

**3. Project Structure**

Initially, our project looks like this:



```
crudexample /home/sssit/cr
    crudexample
        __init__.py
        settings.py
        urls.py
        wsgi.py
    employee
        migrations
            __init__.py
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
    manage.py
External Libraries
```

## 4. Database Setup

Create a database **djangodb** in mysql, and configure into the **settings.py** file of django project. See the example.

**// settings.py**

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'djangodb',
        'USER':'root',
        'PASSWORD':'mysql',
        'HOST':'localhost',
        'PORT':'3306'
    }
}
```

## 5. Create a Model

Put the following code into **models.py** file.

**// models.py**

```
from django.db import models
class Employee(models.Model):
    eid = models.CharField(max_length=20)
    ename = models.CharField(max_length=100)
    eemail = models.EmailField()
    econtact = models.CharField(max_length=15)
    class Meta:
        db_table = "employee"
```

**6. Create a ModelForm**

**// forms.py**

```
from django import forms
from employee.models import Employee
class EmployeeForm(forms.ModelForm):
    class Meta:
        model = Employee
        fields = "__all__"
```

**7. Create View Functions**

**// views.py**

```
from django.shortcuts import render, redirect
from employee.forms import EmployeeForm
from employee.models import Employee
# Create your views here.
def emp(request):
    if request.method == "POST":
        form = EmployeeForm(request.POST)
        if form.is_valid():
            try:
                form.save()
                return redirect('/show')
            except:
                pass
    else:
        form = EmployeeForm()
    return render(request,'index.html',{'form':form})
def show(request):
    employees = Employee.objects.all()
    return render(request,"show.html",{'employees':employees})
def edit(request, id):
    employee = Employee.objects.get(id=id)
    return render(request,'edit.html', {'employee':employee})
def update(request, id):
    employee = Employee.objects.get(id=id)
    form = EmployeeForm(request.POST, instance = employee)
    if form.is_valid():
        form.save()
        return redirect("/show")
    return render(request, 'edit.html', {'employee': employee})
def destroy(request, id):
    employee = Employee.objects.get(id=id)
    employee.delete()
    return redirect("/show")
```

**// urls.py**

```
from django.contrib import admin
from django.urls import path
from employee import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('emp', views.emp),
    path('show',views.show),
    path('edit/<int:id>', views.edit),
    path('update/<int:id>', views.update),
    path('delete/<int:id>', views.destroy),
]
```

## 9. Organize Templates

Create a **templates** folder inside the **employee** app and create three (index, edit, show) html files inside the directory. The code for each is given below.

**// index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Index</title>
{% load staticfiles %}
</head>
<body>
   <form method="POST" action="/emp">
      {% csrf_token %}
      <h3>Enter Details</h3>
      <label>Employee Id:</label>
      {{ form.eid }}
      <br><br>
      <label>Employee Name:</label>
      {{ form.ename }}
      <br><br>
      <label>Employee Email:</label>
      {{ form.eemail }}
      <br><br>
      <label>Employee Contact:</label>
```

```
        {{ form.econtact }}
        <br><br>
        <button type="submit">Submit</button>
    </form>
</body>
```

**// show.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Employee Records</title>
</head>
<body>
    <h2>Employee Records</h2>
    {% for employee in employees %}
        <div>
            <strong>Employee ID:</strong> {{ employee.eid }} <br>
            <strong>Employee Name:</strong> {{ employee.ename }} <br>
            <strong>Employee Email:</strong> {{ employee.eemail }} <br>
            <strong>Employee Contact:</strong> {{ employee.econtact }} <br>
            <a href="/edit/{{ employee.id }}">Edit</a> |
            <a href="/delete/{{ employee.id }}">Delete</a>
        </div>
        <hr>
    {% endfor %}
    <br>
    <center><a href="/emp">Add New Record</a></center>
</body>
</html>
```

**// edit.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Update Employee</title>
</head>
<body>
    <h3>Update Employee Details</h3>
```

```
   <form method="POST" action="/update/{{ employee.id }}">
      {% csrf_token %}
      <label>Employee Id:</label>
      <input type="text" name="eid" required maxlength="20" value="{{ employee.eid }}">
      <br><br>
      <label>Employee Name:</label>
      <input type="text" name="ename" required maxlength="100" value="{{ employee.ename }}">
      <br><br>
      <label>Employee Email:</label>
      <input type="email" name="eemail" required maxlength="254" value="{{ employee.eemail }}">
      <br><br>
      <label>Employee Contact:</label>
      <input type="text" name="econtact" required maxlength="15" value="{{ employee.econtact }}">
      <br><br>
      <button type="submit">Update</button>
   </form>
</body>
</html>
```

**12. Create Migrations**

Create migrations for the created model employee, use the following command.

```
$ python3 manage.py makemigrations
```

**// settings.py**

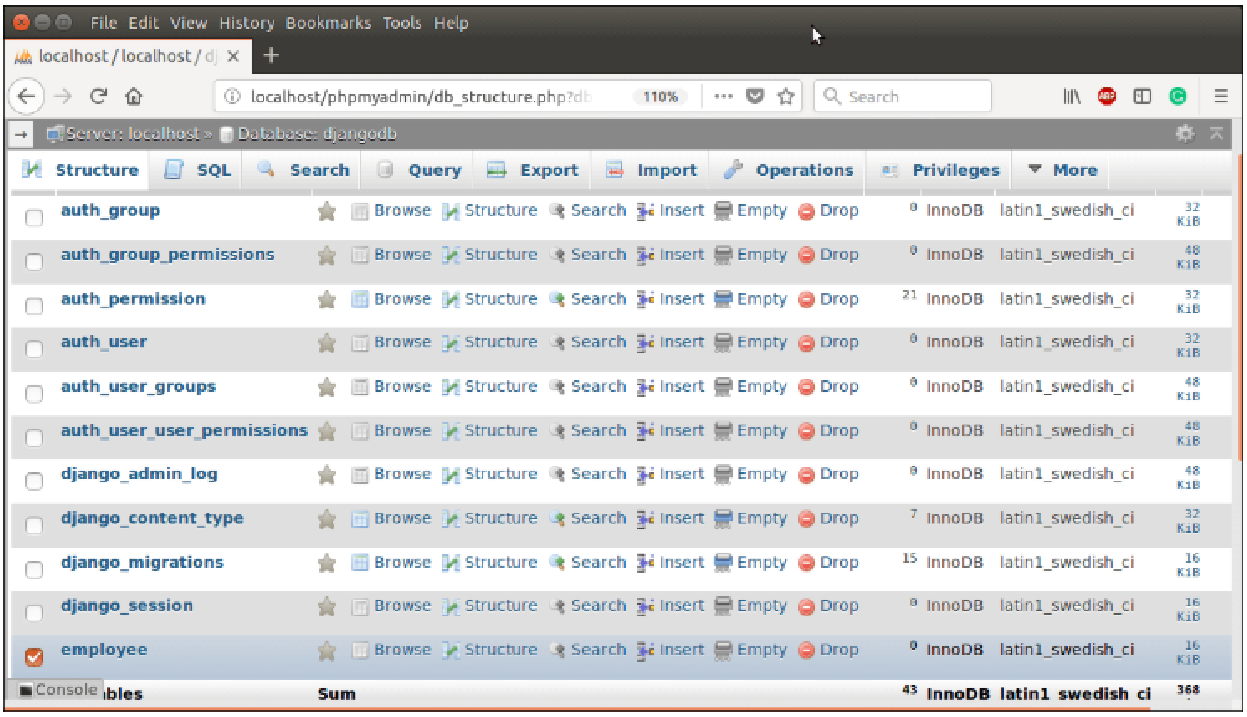
```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'employee'
]
```

Run the command to migrate the migrations.

```
$ python3 manage.py migrate
```

Now, our application has successfully connected and created tables in database. It creates 10 default tables for handling project (session, authentication etc) and one table of our model that we created.

See list of tables created after migrate command.



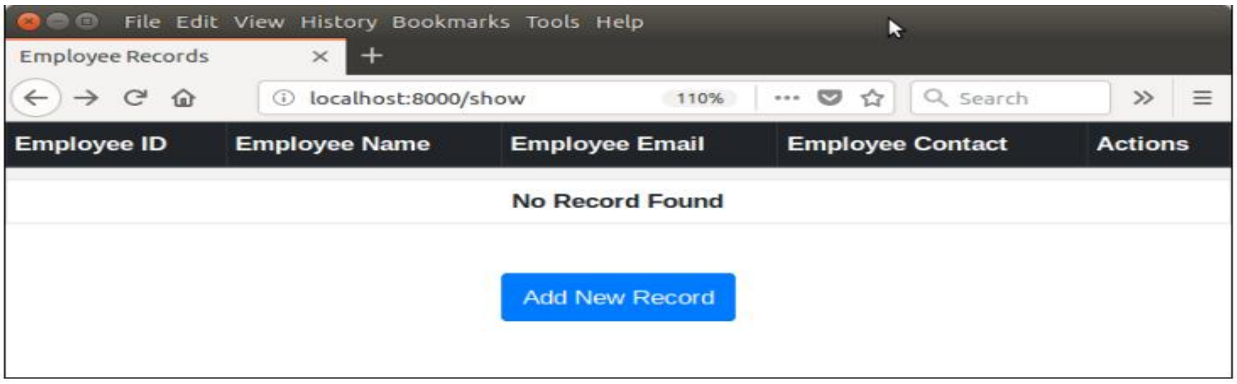
localhost/phpmyadmin/db_structure.php?db

Server: localhost » Database: djangodb

Structure | SQL | Search | Query | Export | Import | Operations | Privileges | More

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ☐ | auth_group | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 0 | InnoDB | latin1_swedish_ci | 32 KiB |
| ☐ | auth_group_permissions | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 0 | InnoDB | latin1_swedish_ci | 48 KiB |
| ☐ | auth_permission | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 21 | InnoDB | latin1_swedish_ci | 32 KiB |
| ☐ | auth_user | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 0 | InnoDB | latin1_swedish_ci | 32 KiB |
| ☐ | auth_user_groups | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 0 | InnoDB | latin1_swedish_ci | 48 KiB |
| ☐ | auth_user_user_permissions | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 0 | InnoDB | latin1_swedish_ci | 48 KiB |
| ☐ | django_admin_log | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 0 | InnoDB | latin1_swedish_ci | 48 KiB |
| ☐ | django_content_type | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 7 | InnoDB | latin1_swedish_ci | 32 KiB |
| ☐ | django_migrations | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 15 | InnoDB | latin1_swedish_ci | 16 KiB |
| ☐ | django_session | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 0 | InnoDB | latin1_swedish_ci | 16 KiB |
| ✓ | employee | ☆ | Browse | Structure | Search | Insert | Empty | Drop | 0 | InnoDB | latin1_swedish_ci | 16 KiB |
| | bles | Sum | | | | | | | 43 | InnoDB | latin1 swedish ci | 368 |

## Run Server

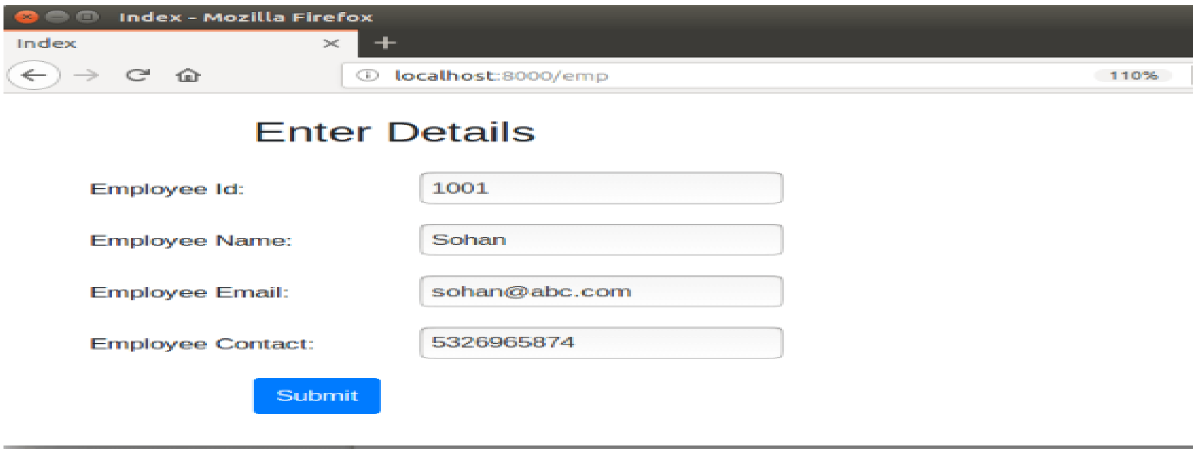


localhost:8000/show

| Employee ID | Employee Name | Employee Email | Employee Contact | Actions |
|---|---|---|---|---|
| | | No Record Found | | |

Add New Record

Filling the details.



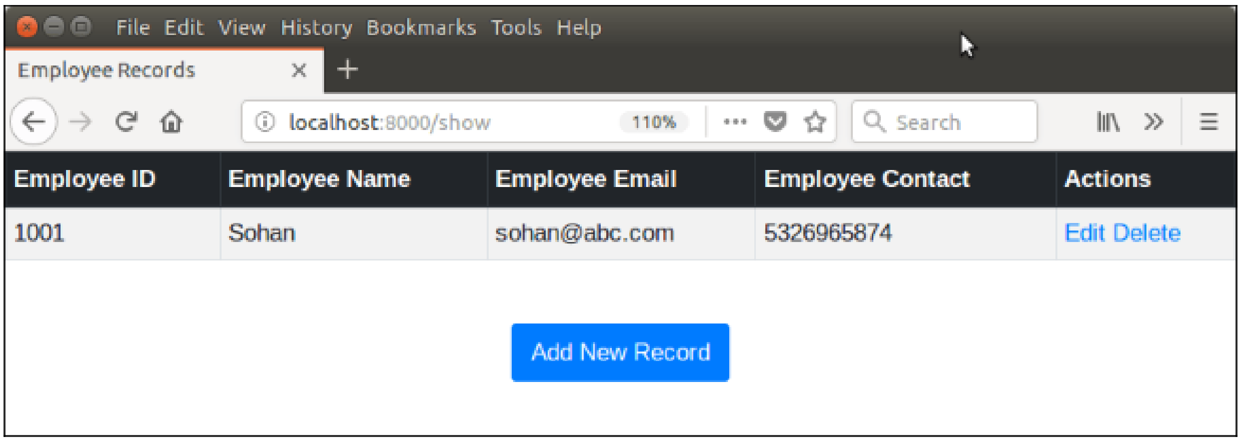
localhost:8000/emp

Enter Details

| | |
|---|---|
| Employee Id: | 1001 |
| Employee Name: | Sohan |
| Employee Email: | sohan@abc.com |
| Employee Contact: | 5326965874 |

Submit

Submit the record and see, after submitting it shows the saved record.

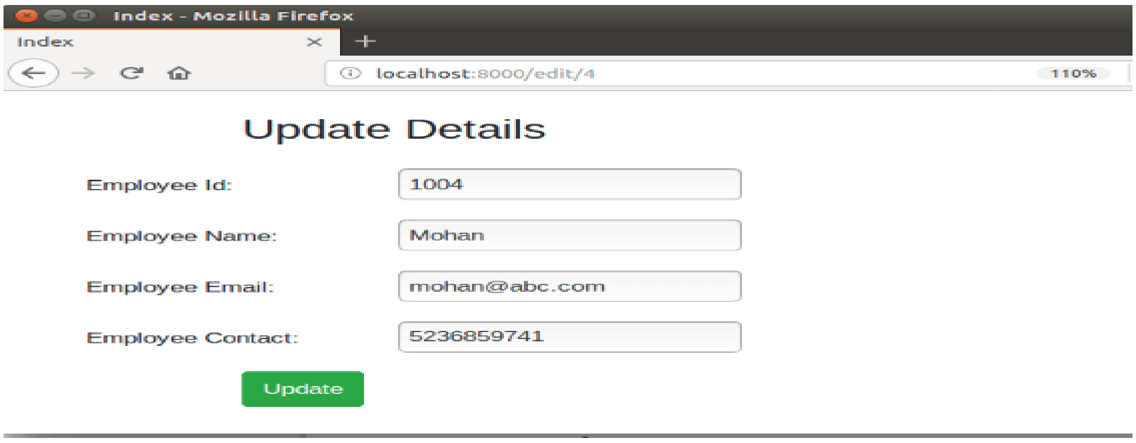


| Employee ID | Employee Name | Employee Email | Employee Contact | Actions |
|---|---|---|---|---|
| 1001 | Sohan | sohan@abc.com | 5326965874 | Edit Delete |

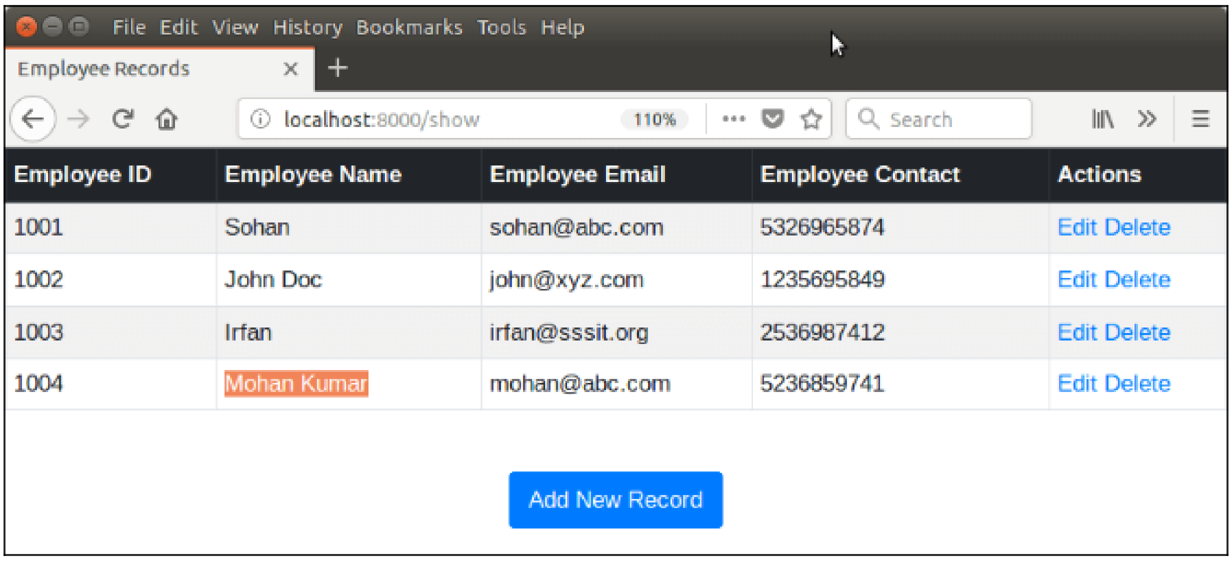
This section also allows, update and delete records from the **actions** column.

## Update Record

Lets update the record of **Mohan** by clicking on **edit** button. It will display record of Mohan in edit mode.



| | |
|---|---|
| Employee Id: | 1004 |
| Employee Name: | Mohan |
| Employee Email: | mohan@abc.com |
| Employee Contact: | 5236859741 |



| Employee ID | Employee Name | Employee Email | Employee Contact | Actions |
|---|---|---|---|---|
| 1001 | Sohan | sohan@abc.com | 5326965874 | Edit Delete |
| 1002 | John Doc | john@xyz.com | 1235695849 | Edit Delete |
| 1003 | Irfan | irfan@sssit.org | 2536987412 | Edit Delete |
| 1004 | Mohan Kumar | mohan@abc.com | 5236859741 | Edit Delete |

Same like, we can delete records too, by clicking the **delete** link.