Chapter-4

Working with Models And Databases

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

Working with Models and Databases:

- As the part of web application development, compulsory we required to interact with database to store our data and to retrieve our stored data.
- Django provides a big in-built support for database operations. Django provides one inbuilt database sqlite3.
- For small to medium applications this database is more enough. Django can provide support for other databases also like oracle, mysql,postgresql etc

Database Configuration:

- Django by default provides sqlite3 database. If we want to use this database,we are not required to do any configurations.
- The default sqllite3 configurations in settings.py file are declared as follows.

settings.py

```
1) DATABASES = {
2)     'default': {
3)         'ENGINE': 'django.db.backends.sqlite3',
4)         'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
5)     }
6) }
```

- If we don't want sqlite3 database then we have to configure our own database with the following parameters.

1) ENGINE: Name of Database engine
2) NAME: Database Name
3) USER: Database Login user name
4) PASSWORD: Database Login password
5) HOST: The Machine on which database server is running
6) PORT: The port number on which database server is running

Note: Most of the times HOST and PORT are optional.

How to Check Django Database Connection:

- We can check whether django database configurations are properly configured or not by using the following commands from the shell
- D:\djangoprojects\modelProject>py manage.py shell
  >>> from django.db import connection
  >>> c = connection.cursor()
- If we are not getting any error means our database configurations are proper.

Configuration of MySQL Database:

- First we have to create our own logical database in the mysql.
  mysql> create database employeedb;
- We have to install mysqlclient by using pip as follows
  pip install --only-binary :all: mysqlclient

settings.py

```
1)  DATABASES = {
2)     'default': {
3)        'ENGINE': 'django.db.backends.mysql',
4)        'NAME': 'employeedb',
5)        'USER':'root',
6)        'PASSWORD':'root'
7)     }
8)  }
```

Checking Configurations:

D:\djangoprojects\modelProject>py manage.py shell

```
>>> from django.db import connection
>>> c = connection.cursor()
```

Configuration of Oracle Database:

```
1)  DATABASES = {
2)     'default': {
3)        'ENGINE': 'django.db.backends.oracle',
4)        'NAME': 'XE',
5)        'USER':'scott',
6)        'PASSWORD':'tiger'
7)     }
8)  }
```

Note: We can find oracle database name by using the following command.

```
SQL> select * from global_name;
```

Model Class:

- A Model is a Python class which contains database information.
- A Model is a single, definitive source of information about our data. It contains fields and behavior of the data what we are storing.

- Each model maps to one database table.
- Every model is a Python class which is the child class of (django.db.models.Model)
- Each attribute of the model represents a database field.
- We have to write all model classes inside 'models.py' file.

1. Create a project and application and link them.

```
django-admin startproject modelProject
cd modelProject/
python manage.py startapp testApp
```

After creating a project and application, in the models.py file, write the following code:

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=30)
7)      esal=models.FloatField()
8)      eaddr=models.CharField(max_length=30)
```

Note: This model class will be converted into Database table. Django is responsible for this.

table_name: appName_Employee
fields: eno, ename, esal and eaddr. And one extra field: id
behaviors: eno is of type Integer, ename is of type Char and max_length is 30 characters.

Hence, Model Class = Database Table Name + Field Names + Field Behaviors

Converting Model Class into Database specific SQL Code:

Once we write Model class, we have to generate the corresponding SQL Code. For this, we have to use "makemigrations" command.
Python manage.py make migrations

It results the following:

```
Migrations for 'testApp':
  testApp/migrations/0001_initial.py
- Create model Employee
```

How to see corresponding SQL Code of Migrations:

To see the generated SQL Code, we have to use the following command "sqlmigrate"
python manage.py sqlmigrate testApp 0001

```
1) BEGIN;
2) --
3) -- Create model Employee
4) --
5) CREATE TABLE "testApp_employee" ("id" integer NOT NULL PRIMARY KEY AUTOIN
   CREMENT, "eno" integer NOT NULL, "ename" varchar(30) NOT NULL, "esal" real N
   OT NULL, "eaddr" varchar(30) NOT NULL);
6) COMMIT;
```

Note: **Here 0001 is the file passed as an argument**

"id" field:

1) **For every table(model), Django will generate a special column named with "id".**
2) **ID is a Primary Key. (Unique Identifier for every row inside table is considered as a primary key).**
3) **This field(id) is auto increment field and hence while inserting data, we are not required to provide data for this field.**
4) **This id field is of type "AutoField"**
5) **We can override the behavior of "id" field and we can make our own field as "id".**
6) **Every Field is by default "NOT NULL".**

How to execute generated SQL Code (migrate Command):

After generating sql code, we have to execute that sql code to create table in database. For this, we have to use 'migrate' command.

python manage.py migrate

Operations to perform:

Apply all migrations: admin, auth, contenttypes, sessions, testApp
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 sessions.0001_initial... OK
- Applying testApp.0001_initial... OK

Note: Now tables will be created in the database.

What is the Advantage of creating Tables with 'migrate' Command

If we use 'migrate' command, then all Django required tables will be created in addition to our application specific tables. If we create table manually with sql code, then only our application specific table will be created and django my not work properly. Hence it is highly recommended to create tables with 'migrate' command.

How to Check created Table in Django admin Interface:

We have to register model class in 'admin.py' file.

admin.py

```
1) from django.contrib import admin
2) from testApp.models import Employee
3)
4) # Register your models here.
5)
6) admin.site.register(Employee)
```

Creation of Super User to login to admin Interface:

- We can create super user with the following command by providing username, mailid, password.

python manage.py createsuperuser

- We can login to admin interface → Start the server and login to admin interface using the created credentials.
- python manage.py runserver
- Open the following in browser: http://127.0.0.1:8000/admin/

Difference between makemigrations and migrate:

"makemigrations" is responsible to generate SQL code for Python model class whereas "migrate" is responsible to execute that SQL code so that tables will be created in the database.

To Display Data in admin Interface in Browser:

models.py

```
1)  from django.db import models
2)
3)  # Create your models here.
4)
5)  class Employee(models.Model):
6)      eno=models.IntegerField()
7)      ename=models.CharField(max_length=30)
8)      esal=models.FloatField()
9)      eaddr=models.CharField(max_length=30)
10)  def __str__(self):
11)         return 'Employee Object with eno: +str(self.no)'
```

admin.py

```
1)  from django.contrib import admin
2)  from testApp.models import Employee
3)
4)  # Register your models here.
5)
6)  class EmployeeAdmin(admin.ModelAdmin):
7)      list_display=['eno','ename','esal','eaddr']
8)
9)  admin.site.register(Employee,EmployeeAdmin)
```

Note:

We should do this registration in a single line otherwise we are getting error.
admin.site.register(Employee)
admin.site.register(EmployeeAdmin)

Practice Activity:

Create the following tables and populate with some sample data

1. **Job(posting date,location,offeredsalary,qualification)**
2. **Book(number,title,author,publisheddate)**
3. **Customer(name,ano,mailid,phonenumber,age)**

Now we can write views to get data from the database and send to template.
Before writing views.py file, create "templates" and "static" folder with respective application folders and HTML and CSS files and link them in settings.py file.

Views.py

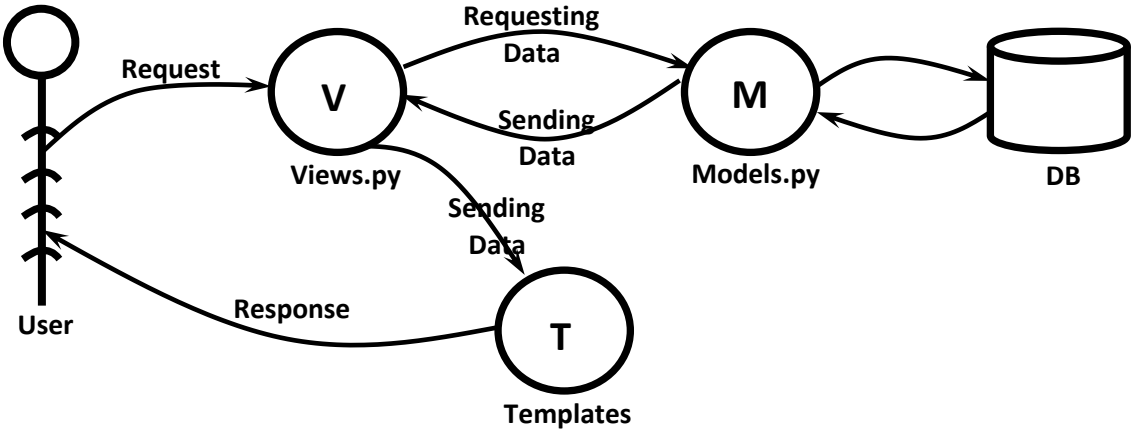
```
1)  from django.shortcuts import render
2)  from testApp.models import Employee
3)
4)  # Create your views here.
5)  def empdata(request):
6)      emp_list=Employee.objects.all()
7)      my_dict={'emp_list':emp_list}
8)      return render(request, 'testApp/emp.html', context=my_dict)
```

emp.html

```
1)  <!DOCTYPE html>
2)  {% load staticfiles %}
3)  <html lang="en" dir="ltr">
4)    <head>
5)      <meta charset="utf-8">
6)      <title></title>
7)      <link rel="stylesheet" href="{% static '/css/demo.css'%}">
8)    </head>
9)
10)  <body>
11)    <h1> The employees list is : </h1>
12)
13)    {% if emp_list %}
14)    <table>
15)      <thead>
16)        <th> eno </th>
17)        <th> ename </th>
18)        <th> esal </th>
19)        <th> eaddr </th>
20)      </thead>
21)
22)      {% for emp in emp_list %}
23)      <tr>
24)        <td> {{emp.eno}}</td>
25)        <td>{{emp.ename}}</td>
26)        <td>{{emp.esal}}</td>
27)        <td> {{emp.eaddr}}</td>
28)      </tr>
29)      {% endfor %}
30)
31)    </table>
32)    {%else%}
```

```
33)   <p> No records found </p>
34)   {% endif %}
35)
36)  </body>
37) </html>
```

MVT Diagram



Important FAQs Related to Model and database configurations:

1) How to configure database inside settings.py?
2) How to check connections?
3) How to define Model class inside models.py
4) How we can perform makemigrations?
5) How we can perform migrate?
6) How to add our model to admin interface inside admin.py
7) To display total data how to write ModelAdmin class inside admin.py
8) How to createsuperuser?
9) How to login to admin interface and add data to our tables?
10) How to see generated sqlcode b'z of makemigrations

Faker Module:

We can use Faker Module to generate fake data for our database models.

```
1) from faker import Faker
2) from random import *
3) fakegen=Faker()
4) name=fakegen.name()
5) print(name)
```

```
first_name=fakegen.first_name()
last_name=fakegen.last_name()
print(first_name)
print(last_name)
date=fakegen.date()
print(date)
number=fakegen.random_number(5)
print(number)
email=fakegen.email()
print(email)
print(fakegen.city())
print(fakegen.random_int(min=0, max=9999))
print(fakegen.random_element(elements=('Project Manager', 'TeamLead', 'Software Engineer')))
```

Working with MYSQL Database (studentinfo project):

settings.py

```
TEMPLATE_DIR=os.path.join(BASE_DIR,'templates')
INSTALLED_APPS = [
    ....,
    'testapp'
]
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATE_DIR],
    ....
    }
]
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'studentdb',
        'USER':'root',
        'PASSWORD':'root'
    }
}
```

models.py

```
1)  from django.db import models
2)
3)  # Create your models here.
4)  class Student(models.Model):
5)      name=models.CharField(max_length=30)
6)      marks=models.IntegerField()
```

admin.py

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

views.py

```
1)  from django.shortcuts import render
2)  from testapp.models import Student
3)
4)  # Create your views here.
5)  def studentview(request):
6)      student_list=Student.objects.order_by('marks')
7)      my_dict={'student_list':student_list}
8)      return render(request,'testapp/students.html',context=my_dict)
```

students.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)      {%if student_list %}
9)      <table border="1" margin='auto'>
10)       <thead>
11)         <th>Name</th>
12)         <th>Marks</th>
```

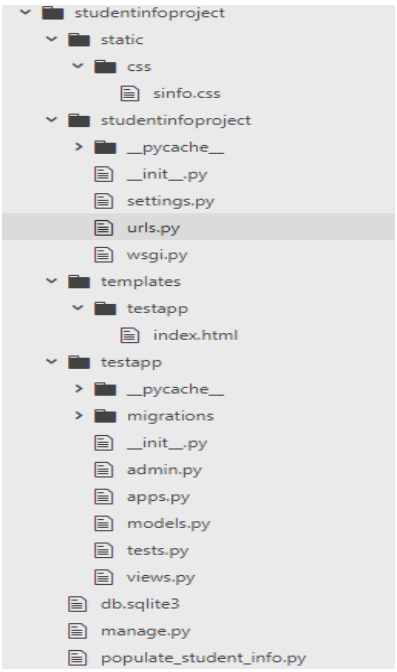
```
    </thead>
    {% for student in student_list %}
    <tr>
      <td>{{student.name}}</td>
      <td>{{student.marks}}</td>
    </tr>
    {% endfor %}
  </table>
  {% else %}
  <p>No Studetns found</p>
{% endif %}
 </body>
</html>
```

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'^$', views.studentview),
]
```

Working with MYSQL Database and Faker Module (studentinfo Project):



settings.py

```
1) TEMPLATE_DIR=os.path.join(BASE_DIR,'templates')
2) STATIC_DIR=os.path.join(BASE_DIR,'static')
3) INSTALLED_APPS = [
4)     ....,
5)     'testapp'
6) ]
7) TEMPLATES = [
8)     {
9)         'BACKEND': 'django.template.backends.django.DjangoTemplates',
10)        'DIRS': [TEMPLATE_DIR],
11)    ....
12)    }
13) ]
14) DATABASES = {
15)    'default': {
16)        'ENGINE': 'django.db.backends.mysql',
17)        'NAME': 'studentdb',
18)        'USER':'root',
19)        'PASSWORD':'root'
20)    }
21) }
22) STATIC_URL = '/static/'
23) STATICFILES_DIRS=[
24) STATIC_DIR,
25) ]
```

models.py

```
1) from django.db import models
2)
3) # Create your models here.
4) class Student(models.Model):
5)     rollno=models.IntegerField()
6)     name=models.CharField(max_length=30)
7)     dob=models.DateField()
8)     marks=models.IntegerField()
9)     email=models.EmailField()
10)    phonenumber=models.IntegerField(15)
11)    address=models.TextField()
```

admin.py

```
1)  from django.contrib import admin
2)  from testapp.models import Student
3)
4)  # Register your models here.
5)  class StudentAdmin(admin.ModelAdmin):
6)      list_display=['rollno','name','dob','marks','email','phonenumber','address']
7)
8)  admin.site.register(Student,StudentAdmin)
```

populate_student_info.py

```
1)  import os
2)  os.environ.setdefault('DJANGO_SETTINGS_MODULE','studentinfoproject.settings')

3)  import django
4)  django.setup()
5)
6)  from testapp.models import Student
7)  from faker import Faker
8)  from random import *
9)  fake=Faker()
10) def phonenumbergen():
11)     d1=randint(6,9)
12)     num=''+str(d1)
13)     for i in range(9):
14)         num=num+str(randint(0,9))
15)     return int(num)
16) def populate(n):
17)     for i in range(n):
18)         frollno=fake.random_int(min=1,max=999)
19)         fname=fake.name()
20)         fdob=fake.date()
21)         fmarks=fake.random_int(min=1,max=100)
22)         femail=fake.email()
23)         fphonenumber=phonenumbergen()
24)         faddress=fake.address()
25)         student_record=Student.objects.get_or_create(rollno=frollno,name=fname,do
    b=fdob,marks=fmarks,email=femail,phonenumber=fphonenumber,address=faddre
    ss)
26) populate(30)
```

views.py

```
1)  from django.shortcuts import render
2)  from testapp.models import Student
3)
4)  # Create your views here.
5)  def home_page_view(request):
6)      students=Student.objects.all()
7)      #students=Student.objects.filter(marks__lt=35)
8)      #students=Student.objects.filter(name__startswith='A')
9)      #students=Student.objects.all().order_by('marks')
10)     #students=Student.objects.all().order_by('-marks')
11)     return render(request,'testapp/index.html',{'students':students})
```

index.html

```
1)  <!DOCTYPE html>
2)  {%load staticfiles%}
3)  <html lang="en" dir="ltr">
4)    <head>
5)      <meta charset="utf-8">
6)      <title></title>
7)      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7
    /css/bootstrap.min.css" integrity="sha384-
    BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" c
    rossorigin="anonymous">
8)      <link rel="stylesheet" href="{%static "css/sinfo.css"%}">
9)    </head>
10)   <body>
11)     <div class="container" >
12)       <h1>Student Information</h1><hr>
13)       {% if students%}
14)         {%for student in students%}
15)           <h2>{{student.name}} Information</h2><hr>
16)           <ul>
17)             <li>Student Rollno:{{student.rollno}}</li>
18)             <li>Student DOB:{{student.dob}}</li>
19)             <li>Student Marks:{{student.marks}}</li>
20)             <li>Student Email:{{student.email}}</li>
21)             <li>Student Phone Number:{{student.phonenumber}}</li>
22)             <li>Student Address:{{student.address}}</li>
23)           </ul>
24)           <br>
25)         {%endfor%}
26)       {%else%}
```

```
27)       <p>Student Records Are Not Available</p>
28)     {%endif%}
29)    </div>
30)  </body>
31) </html>
```

sinfo.css

```
1)  body .container{
2)    background: red;
3)    color:white;
4)  }
5)  h1{
6)    text-align: center;
7)  }
8)  h2{
9)    color:yellow;
10) }
```

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.home_page_view),
8)  ]
```