Django Web Framework

Django Models and Databases

Necessity of Databases in Web Development-

In web development, databases are essential for storing, managing, and retrieving data efficiently. They allow dynamic content generation, user authentication, transaction management, and data consistency. Without databases, web applications would be static and unable to handle the complex, data-driven needs of modern users, such as user profiles, product listings, or real-time updates.

For example- The data you upload on social media is stored somewhere in the database, same way the data you download is also stored somewhere in a database.

User uses web browser to send a request and that request is then handled by a web server. Web server interacts with the database and then provide appropriate response.

Django Models and Databases-

Django simplifies the creation and management of databases through its powerful Object-Relational Mapping (ORM) system.

What are Models?

Django models are Python classes that define the structure of your database tables. Each attribute in the model corresponds to a database field.

We define Django models in models.py file inside our Django applications.

Default Django database-

Django by default provides sqlite3 database. If we want to use this database, we are not required to do any configurations.

When you start a new Django project, the default database configured is SQLite.

SQLite's simplicity and ease of use make it the ideal default database for Django, especially during development. However, as your application grows and requires more robust database features, Django's ORM makes it straightforward to transition to a more powerful database system.

For small to medium applications this database is more enough. Django can provide support for other databases also like oracle, MySQL , PostgreSQL etc.

When we create a new project-

```
C:\Users\LENOVO\Documents\djangocourse>django-admin startproject modelProject1
```

The default database configurations are already added to the configuration file-settings.py.

```
76  DATABASES = {
77      'default': {
78          'ENGINE': 'django.db.backends.sqlite3',
79          'NAME': BASE_DIR / 'db.sqlite3',
80      }
81  }
```

If we want to use any other database, then we have to change the configurations for databases in settings.py file.

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

Some of the above parameters are optional.

How to configure MySQL database -

- Firstly, we have to create a database in the MySQL database. You can use MySQL command-line client installed on your local machine.
- Then create a database. Example-

```
mysql> create database studentdb;
```

- For Django to communicate with MySQL, you need to install the mysqlclient library. You can do this using pip:

```
pip install mysqlclient
```

- Now create a project in Django.

```
C:\Users\LENOVO\Documents\djangocourse>django-admin startproject modelProject1
```

- Open settings.py file in IDE and change the database configurations to:

```
DATABASES={
    'default':{
        'ENGINE':'django.db.backends.mysql',
        'NAME':'studentdb',
        'USER':'your_user_name',
        'PASSWORD':'your_password'
    }
}
```

In settings.py we have to provide the -

- ENGINE- django.db.backends.mysql
- NAME- Name of the database
- USER- Your mysql Username
- PASSWORD- Your mysql password
- HOST- localhost or any other (optional)
- PORT- 3306 (Optional) #Default MySQL port

Models-

Django models are Python classes that define the structure of your database tables. Each attribute in the model corresponds to a database field.

We define Django models in models.py file inside our Django applications.

In Django, a model is a Python class that represents a database table. Each model class corresponds to a table in the database, and each attribute of the class represents a column in that table. The fields in the model class define the data types and constraints of the columns in the database.

Django models allow you to interact with your database using Python code instead of writing raw SQL queries. The Django ORM (Object-Relational Mapping) automatically handles the conversion between Python objects and database records.

Example of model class defined inside models.py file-

```
modelapp > models.py > Author
1   from django.db import models
2
3   # Create your models here.
4   class Author(models.Model):
5       author_name=models.CharField(max_length=35)
6       birthdate=models.DateField()
7       city=models.CharField(max_length=20,default='Mumbai')
8
9       def __str__(self):
10          return self.author_name
```

In this example:

- The Author class is a model that represents a table in the database.
- The name, city and birthdate attributes are fields that correspond to columns in the Author table.
- In the context of Django models, the __str__ function (or __str__ method) is a special method that is used to define a human-readable string representation of an object. When you define the __str__ method in your Django model, it controls what will be displayed when you print an instance of that model or when it is displayed in the Django admin interface, the Django shell, or other contexts where the object is converted to a string.

How It Works

- **Without __str__**: If you don't define a __str__ method, Django will fall back to a default string representation, which is usually something like <Author: Author object (1)>, where 1 is the primary key of the object.
- **With __str__**: When you define the __str__ method, and you have an Author object with the name "Harsh", printing this object or displaying it in the Django admin will show "Harsh" instead of the less informative default.

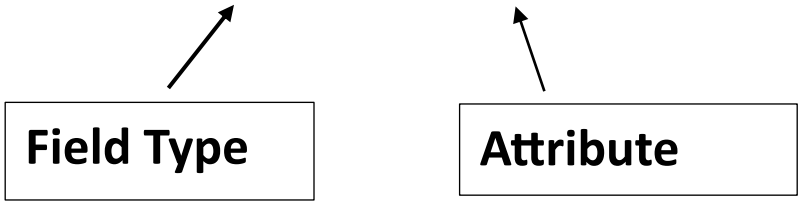
Different Types of Fields in Django Models

Django provides a variety of field types to define the structure and constraints of your model's data. Here are some of the most commonly used field types:

1. CharField-

- **Description**: A field for storing strings of a fixed length.

```
name = models.CharField(max_length=100)
```



2. TextField-

- **Description**: A field for storing large text. No maximum length is enforced.

```
description = models.TextField()
```

3. IntegerField-

- **Description**: A field for storing integer values.

```
age = models.IntegerField()
```

4. FloatField-

- **Description**: A field for storing floating-point numbers.

```
price = models.FloatField()
```

5. DecimalField-

- **Description**: A field for storing decimal numbers with fixed precision.

```
price = models.DecimalField(max_digits=10, decimal_places=2)
```

- **Attributes**:
- max_digits: The maximum number of digits allowed.
- decimal_places: The number of decimal places to store.

6. BooleanField-

- **Description**: A field for storing True or False values.

```
is_active = models.BooleanField(default=True)
```

7. DateField-

- **Description**: A field for storing date values.

```
birthdate = models.DateField()
```

8. DateTimeField-

- **Description: A field for storing date and time values.**

```
created_at = models.DateTimeField(auto_now_add=True)
```

- **Attributes**:
- auto_now_add: Automatically sets the field to the current date/time when the object is first created.

- auto_now: Automatically updates the field to the current date/time every time the object is saved.

9. EmailField-

- **Description**: A field for storing email addresses.

```
email = models.EmailField()
```

10. **URLField-**

- **Description**: A field for storing URLs.

```
website = models.URLField()
```

11. FileField-

- **Description**: A field for uploading and storing files.

```
file = models.FileField(upload_to='uploads/')
```

12. ImageField-

- **Description**: A field for uploading and storing images (inherits from FileField).

```
image = models.ImageField(upload_to='images/')
```

NOTE: When we learn about ORM we will discuss Foreign Key and other fields.

Migrations-

Django uses migrations to propagate changes you make to your models (like adding a new field) into your database schema. The makemigrations command creates new migration files based on the changes in your models, and the migrate command applies these migrations to the database.

Django migrations are a powerful tool to manage the evolution of your database schema over time. They essentially track changes made to your models and automatically generate the necessary SQL code to apply those changes to your database. This ensures that your database structure always aligns with your models.

How Migrations Work

1. **Model Changes:** You make changes to your models, such as adding a new field, renaming a field, or deleting a model.
2. **makemigrations:** This command analyzes your models and creates a migration file in the migrations directory of your app. This file contains the changes you made.
3. **migrate:** This command applies the changes defined in the migration file to your database.

Let's see a practical example to understand the migrations-

Step 1: We already created a project named modelProject1

Step 2: Now lets create a application named modelApp inside the above project and configure it in settings.py file.

```
C:\Users\LENOVO\Documents\djangocourse\modelProject1>python manage.py startapp modelApp
```

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

Step3: create model class inside our models.py file. Changing database configuration inside our settings.py file or keeping the default database sqlite3.

For this project we are using the default database configuration for sqlite3.

modelapp > models.py > Author

```python
from django.db import models

# Create your models here.
class Author(models.Model):
    author_name=models.CharField(max_length=35)
    birthdate=models.DateField()
    city=models.CharField(max_length=20,default='Mumbai')

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

Step 4: Apply and run migrations-

After creating a model, we have to run migrations in our terminal.

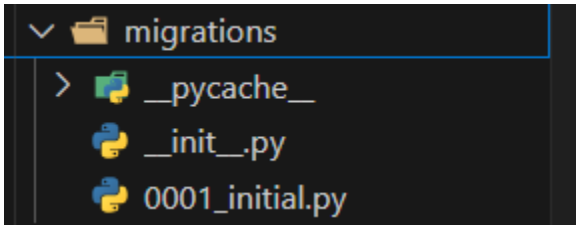
Command for makemigrations and migrate-

- **python manage.py makemigrations**
- **python manage.py migrate**

How it works:

1. **makemigrations**:
    - Examines your models for changes.

- Creates a migration file in your app's migrations directory.
- This migration file contains Python code that describes the changes to be made.



2. **migrate**:
   - Reads the migration files.
   - Translates the Python-based changes into SQL statements.
   - Executes these SQL statements on your database.


```
C:\Users\LENOVO\Documents\djangocourse\modelProject1>python manage.py makemigrations
No changes detected

C:\Users\LENOVO\Documents\djangocourse\modelProject1>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, modelApp, 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 admin.0003_logentry_add_action_flag_choices... 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 auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying modelApp.0001_initial... OK
  Applying sessions.0001_initial... OK
```

NOTE: After every change in models.py file, you have to run both the above commands.

Step 5: Create a view to add data in our table and then display it.

```
modelapp > views.py > ...
1  from django.shortcuts import render
2  from modelapp.models import Author
3  from django.http import HttpResponse
4  # Create your views here.
5
6  def strview(request):
7      author=Author(author_name="Harsh",birthdate="2000-12-14")
8      author.save()
9      object_data=author
10     print(object_data)
11     return HttpResponse(object_data)
```

Import our model from models.py file

The above function strview() is taking values for our fields name and channel_name and then printing it on our console and web page.

In the Django admin site or shell:

Instead of <Author: Author object (1)>

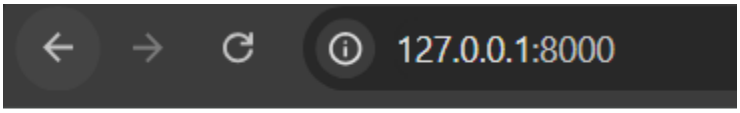
It will display: Harsh

Step 6: Create URL for our view in urls.py

```
from django.contrib import admin
from django.urls import path
from modelApp import views

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

Step 7: Run development server and visit the website.

```
C:\Users\LENOVO\Documents\djangocourse\modelProject1>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
August 09, 2024 - 19:23:51
Django version 5.0, using settings 'modelProject1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

Harsh
[09/Aug/2024 19:23:54] "GET / HTTP/1.1" 200 5
```



Harsh

In the next chapter we will discuss-

- How to register and view Django models in admin interface.
- How to create superuser
- Faker module to populate sample data and Practical.

If you are following the course series and learning the concepts easily, please subscribe to our YouTube channel. This will motivate us to create more educational, course videos and study material.

Join our growing community of tech enthusiasts! Subscribe to our YouTube channel, where we simplify programming languages in the easiest way possible. Gain a deeper understanding with our clear explanations and receive exclusive notes to enhance your learning journey. Don't miss out on valuable insights – hit that subscribe button now and embark on a programming adventure with us!

Subscribe to our channel:
https://www.youtube.com/@HTEASYLEARNING