8.Django-ORM

What is Django ORM

Orm stands for an Object-relational mapper. The Object-Relational Mapper (ORM) is one of Django's most powerful features, allowing you to interact with your database in the same way that you would with SQL.

Django makes it easy for us to interact with database models, i.e. modify, delete, add and query objects.

Orm is for interacting with the database and plays a role like a bridge between the database and our cod**e.**

Example:

Django

```
from django.db import models


class Employe(models.Model):
    name = models.CharField(max_length = 250)
    country = models.CharField(max_length = 150)
    birth_year = models.IntegerField()
    genre = models.CharField(max_length = 150)
```

Django shell

- **It allows us to save, update and retrieve our models in our code. Interacting with databases while working with projects in production is very essential and unavoidable.**
- **The key benefit of ORMs is their ability to develop quickly. ORMs increase the portability of a project.**
- **If we use ORMs, changing the database is a lot easier.**
- **We can access the ORM by running Python manage.py shell from the main directory of our Django project.**

```
//python manage.py shell
```

import our artist model in the shell in order to use it and interact with it.

```
>>> from musiclib.models import Artist
```

This .models refers to the models.py file inside our app and the artist is the class name of our model.

//retrieve all objects

Article.objects.all()

//fetching some records only

Artist.objects.all()[0].name

Str() function

This is a built-in function in Django that defines how an instance of a model looks both in the admin section and also in the shell which we just did it.

this function takes in a self parameter which instance of the Artist and then returns the name of self.

```
 def __str__(self):
       return self.name
class Employee(models.Model):
   name = models.CharField(max_length = 250)
   country = models.CharField(max_length = 150)
   birth_year = models.IntegerField()
```

```
    genre = models.CharField(max_length = 150)


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

Filter the data

This filter() function returns a string version of the name of any instances of this model both the ones that have been created previously or will be created in the future when we retrieve it.

Save the code and run shell again:

```
Djangofrom musiclib.models import Artist
Artist.objects.all()
```

Django also provides some function to filter data like filter(), exclude(),get(). The filter() function returns a QuerySet with objects that match the lookup parameters passed in. for example:

```
>>> Artist.objects.filter(country='USA')
<QuerySet [<Artist: michael jackson>, <Artist: Kurt Cobain>]>
>>>
```

By passing the country parameter to the filter() we tell Django to give us the artists that their country is saved USA in the database.

The filter function is case-sensitive. For example, if the country field is in capital letters, you need to pass it to filter in capital and if not you may get an error.

You can filter a model by passing each field of the model as a parameter into the function filter. In our case, they are name, country, genre, birth year. Also, you can pass one, two, or more parameters

at the same time for example:

```
>>> Artist.objects.filter(country='England',genre="rock")
>>> Artist.objects.exclude(genre="rock")
```

Modifying the objects

Sometimes we want to modify and change the objects of a model. for example we want to change the michael jackson genre to rock

we can do this as follows:

```
>>> artist = Artist.objects.get(name='michael jackson')
>>> artist.genre= "rock"
>>> artist.save()
```

Deleting the objects

we can also delete instances of the model. For example, we want to delete the artists with the pop genre.

```
>>> artist =  Artist.objects.get(name='Roger Waters')
>>> artist.delete()
```

If you retrieve all the objects:

```
>>> Artist.objects.all()
```

Create a new instance

```
artist1 = Artist()
artist.save()
```