1. **Introduction to Django Models :** ORM, Model

ORM

Object-Relational Mapper is a programming technique that helps application to interact with database such as SQLite, MySQL, PostgreSQL, Oracle.

- create a database schema from defined classes or models.
- generate SQL from Python code for a particular database which means developer do not need to write SQL Code.
- helps to change the database easily
- use connectors to connect databases with a web application.

ORM

```
class Student(models.Model):
    stuid=models.IntegerField()
    stuname=models.CharField(max_length=70)
    stuemail=models.EmailField(max_length=70)
    stupass=models.CharField(max_length=70)
```

```
CREATE TABLE "enroll_student" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "stuid" integer NOT NULL,
    "stuname" varchar(70) NOT NULL,
    "stuemail" varchar(70) NOT NULL,
    "stupass" varchar(70) NOT NULL
);
```

id	stuid	stuname	stuemail	stupass

QuerySet

A QuerySet can be defined as a list containing all those objects we have created using the Django model.

QuerySets helps us

- read the data from the database
- filter it
- order it.

Model

A model is the single, definitive source of information about our data.

It contains the

- essential fields and behaviors of the data.
- each model maps to a single database table.

Model Class

- Model class is a class which will represent a table in database.
- Each model is a Python class that subclasses django.db.models.Model
- Each attribute represents a database field.
- Django gives automatically-generated database-access API
- Django provides sqlite database by default.
- We can use other database like MySQL, Oracle SQL etc.

Model Class

```python
from django.db import models

# Create your models here.

class Movie(models.Model):
    movie_title = models.CharField(max_length=150)
    release_year = models.IntegerField()
    director = models.CharField(max_length=100)
    movie_plot = models.TextField()
```

Migrations

Migrations are way of propagating changes to make models (adding a field, deleting a model, etc.) into your database schema.

makemigrations : is used convert model class into sql statements. create a file which will contain sql statements. This file is located in Application's migrations folder.

python manage.py makemigrations

migrate : is used to execute sql statements generated by makemigrations

python manage.py migrate

showmigrations : This lists a project's migrations

Built-in Field Options

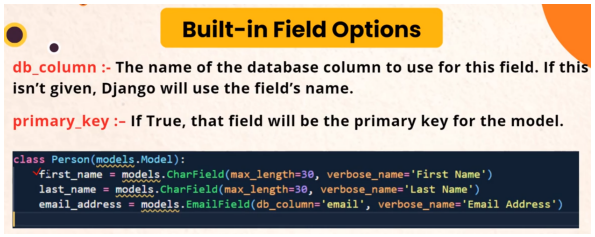
null :- contain either True or False. If True, Django will store empty values as NULL in the database. Default is False.

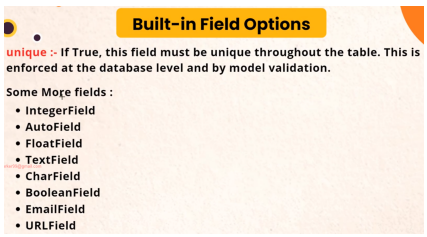
blank :- contain either True or False. If True, the field is allowed to be blank.

Note : *null is purely database-related, whereas blank is validation-related.*

default :- default value for the field.

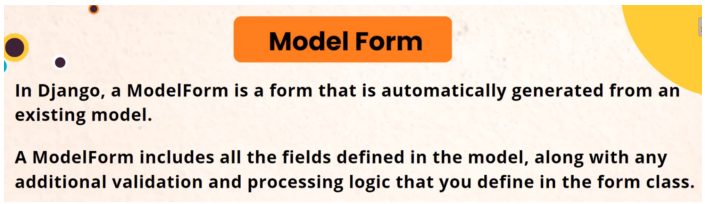
verbose_name :- A human-readable name for the field. If the verbose name isn't given, Django will automatically create it using the field's attribute name, converting underscores to spaces.





Schema → a Complete Database
QuerySet → collection of model's dataset
Model → essential fields and behaviors of the data
Model Class → Python Class, attributes = fields
Migrations → formal approach to let Django convert python into SQL by ORM
Makemigrations → **convert** model class into SQL statements
Migrate → to **execute** / run SQL statements

2. **Creating a Model in Django :**
3. **Showing Model Data using Vs code and DB Browser :**
4. **Creating Superuser and Accessing Admin Pane :**
   py manage.py createsuperuser
5. **Showing Model data to frontend :**
6. **Deleting Model Data From Frontend :**
7. **Introduction to Model Form in Django :**
   Form → i) Form API ii) Model Form



Model Form

```python
from django.db import models

class Article(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
    content = models.TextField()
```

models.py

```python
from django import forms
from .models import Article

class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        fields = ['title', 'author', 'content']
```

forms.py

Meta Class

a metaclass is a class that defines how other classes should behave. provides additional information about the model form. Here are some common options that can be defined in the Meta class:

model: The model that the form is based on.
fields: A list of fields to include in the form. If this option is not specified, all fields in the model will be included in the form.
exclude: A list of fields to exclude from the form.
widgets: A dictionary of field names and their associated widgets.
labels: A dictionary of field names and their associated labels.
help_texts: A dictionary of field names and their associated help text.
error_messages: A dictionary of field names and their associated error messages.

Meta Class

```python
from django import forms
from .models import Article

class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        fields = ['title', 'author', 'content']
        labels = {
            'title': 'Title of the article',
            'author': 'Author name',
            'content': 'Content of the article'
        }
        help_texts = {
            'title': 'Enter a descriptive title for the article',
            'author': 'Enter the name of the author',
            'content': 'Enter the content of the article'
        }
```

Model Form Fields

Model Field	Form Field
AutoField	Not Represented in the Form
BigAutoField	Not Represented in the Form
BigIntegerField	IntegerField with min_value set to -9223372036854775808 and max_value set to 9223372036854775807.
BinaryField	CharField, if editable is set to True on the model field, otherwise not represented in the form.
BooleanField	BooleanField, or NullBooleanField if null=True.
CharField	CharField with max_length set to the model field's max_length and empty_value set to None if null=True.
DateField	DateField
DateTimeField	DateTimeField
DecimalField	DecimalField
DurationField	DurationField

Model Form Fields

Model Field	Form Field
EmailField	EmailField
FileField	FileField
FilePathField	FilePathField
FloatField	FloatField
ForeignKey	ModelChoiceField
ImageField	ImageField
IntegerField	IntegerField
IPAddressField	IPAddressField
GenericIPAddressField	GenericIPAddressField
ManyToManyField	ModelMultipleChoiceField
NullBooleanField	NullBooleanField
PositiveIntegerField	IntegerField

Model Form Fields

Model Field	Form Field
PositiveSmallIntegerField	IntegerField
SlugField	SlugField
SmallAutoField	Not represented in the form
SmallIntegerField	IntegerField
TextField	CharField with widget=forms.Textarea
TimeField	TimeField
URLField	URLField
UUIDField	UUIDField

save()

save (commit=False/True) Method form. This method creates and saves a database object from the data bound to the form. by default True

save (commit=False/True)
if commit=False , then it will return an object that hasn't yet been saved to the database.

8. **Creating form using Model Form :** models.py, forms.py
9. **Saving Model Form data to Database :** showing form in admin panel.