Relationships in Django are defined in models using **`OneToOneField`**, **`ForeignKey`**, and **`ManyToManyField`**.

---

1. One-to-One Relationship (`OneToOneField`)

- Each record in **Model A** is linked to exactly **one record** in **Model B**.
- Common use case: Extending the built-in Django `User` model.

Example: User and Profile

```
from django.db import models
from django.contrib.auth.models import User

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.TextField()
    birth_date = models.DateField(null=True, blank=True)

    def __str__(self):
        return f"{self.user.username}'s Profile"
```

Explanation:

- Each `User` has exactly **one** `Profile`.
- `on_delete=models.CASCADE` means if the `User` is deleted, the `Profile` is also deleted.

Usage in Django Shell:

```
from django.contrib.auth.models import User
from myapp.models import Profile

user = User.objects.create(username="john")
profile = Profile.objects.create(user=user, bio="Hello world!")
print(profile.user.username)  # Output: john
```

---

2. One-to-Many Relationship (`ForeignKey`)

- Each record in **Model A** can be linked to **multiple records** in **Model B**, but each record in Model B belongs to **one record** in Model A.
- Common use case: Blog posts and authors.

Example: Author and Post

```
class Author(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    author = models.ForeignKey(Author, on_delete=models.CASCADE,
related_name="posts")

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

Explanation:

- One `Author` can write **many posts**.
- `related_name="posts"` lets us access all posts of an author: `author.posts.all()`

Usage in Django Shell:

```
author = Author.objects.create(name="Alice")
post1 = Post.objects.create(title="Post 1", content="Content 1",
author=author)
post2 = Post.objects.create(title="Post 2", content="Content 2",
author=author)

print(author.posts.all())  # Output: <QuerySet [post1, post2]>
```

---

3. Many-to-Many Relationship (`ManyToManyField`)

- Each record in **Model A** can relate to **multiple records** in Model B, and vice versa.
- Common use case: Students and courses.

Example: Student and Course

```
class Course(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Student(models.Model):
    name = models.CharField(max_length=100)
    courses = models.ManyToManyField(Course, related_name="students")

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

Explanation:

- A `Student` can enroll in **many courses**, and a `Course` can have **many students**.
- Django automatically creates a **join table** behind the scenes.

Usage in Django Shell:

```
course1 = Course.objects.create(name="Math")
course2 = Course.objects.create(name="Physics")

student = Student.objects.create(name="Bob")
student.courses.add(course1, course2)

print(student.courses.all())  # Output: <QuerySet [Math, Physics]>
print(course1.students.all())  # Output: <QuerySet [Bob]>
```

Summary Table of Relationships

Relationship	Django Field	Example Use Case
One-to-One	`OneToOneField`	User → Profile
One-to-Many	`ForeignKey`	Author → Blog Posts
Many-to-Many	`ManyToManyField`	Student ↔ Courses