Many-To-Many Relationship:

- If we want Many-To-Many Relationship, then multiple child records can depend on multiple parent records
- For example, multiple Courses like Python, Django are joined by single Student
- Like Virat and multiple Students like Virat, Rohit are joined single Course like Python.
- We use ManyToManyField() to implement Many-To-Many relationaship.
- NOTE : Django creates a new model to store the relationaships of child and parent models internally. This new model is called "**intermediate**" model.
- So after executing the makemigrations and migrate commands then we are getting 13 tables here.
- By default django creates 10 tables for every application and 2 tables for our course model and student models and 1 table is for storing relationships links of the both child and parent tables.

  Syntax : student = models.ManyToManyField(Student)

- Many-To-Many Relationship doesn't support all cascading rules.
- It supports only default cascading rule that is **models.CASCADE**
- If we delete any parent record which has the child record, then corresponding child record will not delete in the child table, it will **delete the relationaships link** from intermediate table.
- If we delete all parent records then all relationships will be deleted from the intermediate table, no child record will be deleted.

Create a project using Many-To-Many-Relationship :

Step1: Create a Django ProjectName like **ManyToMany_Project**

Step2: Create a Application Name like **ManyToMany _App**

Step3: Create Database Name like **7am_mtmdb**

Step4: Goto settings.py file and configure database details under DATEBASE section.

```
DATABASES = {
   'default': {
      'ENGINE'  :  'django.db.backends.mysql',
      'NAME' : '7am_mtmdb',
      'USER' : 'root',
        'PASSWORD' : 'root',
    }
}
```

Step5: Open **models.py** file and create required models

```
from django.db import models

class Student(models.Model):
    sno = models.IntegerField()
    sname = models.CharField(max_length=30)
    marks = models.IntegerField()

    def __str__(self):
        return self.sname

class Course(models.Model):
    cno = models.IntegerField()
    cname = models.CharField(max_length=30)
    cfee = models.FloatField()

    student = models.ManyToManyField(Student)

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

Step7: open **admin.py** file and create required admin logics

```
from django.contrib import admin
from OneToOne_App.models import Student,Course

class StudentAdmin(admin.ModelAdmin):
    list_display = ['sno', 'sname', 'marks']


class CourseAdmin(admin.ModelAdmin):
    list_display = ['cno', 'cname', 'cfee']

admin.site.register(Student, StudentAdmin)
admin.site.register(Course, CourseAdmin)
```

Step8: Execute the makemigrations command to convert model code into SQL code format

```
python  manage.py   makemigrations
```

Step9: Execute the migrate command to execute SQL code in database site and creating tables more models.

```
python  manage.py   migrate
```

Step10: Execute the createsuperuser command for creating admin creadentials.

```
python  manage.py   createsuperuser
```

Then it will ask like below details,

```
Username:   Virat
Email : virat@gmail.com
Password:    admin123
Password (again):  admin123
```

Step11: Now execute the runserver command for running the project

```
python  manage.py   runserver   8000
```

Step12: Now open the required browser and then send admin/ url request from the browser then we will get admin login page response like below

Now we will see our Student and Course model related tables in Admin site.

Now add some records into both Student and course tables and open them.

Step13: Goto database and check the tables

```
mysql> select * from manytomany_app_student;
+----+-----+-------+-------+
| id | sno | sname | marks |
+----+-----+-------+-------+
|  1 | 101 | Virat |    80 |
|  2 | 102 | Rohit |    90 |
|  3 | 103 | Surya |    85 |
+----+-----+-------+-------+
mysql> select * from manytomany_app_course;
```

```
+----+-----+---------+------+
| id | cno | cname   | cfee |
+----+-----+---------+------+
|  1 | 201 | Python  | 5000 |
|  2 | 202 | Django  | 6000 |
|  3 | 203 | RESTAPI | 3000 |
+----+-----+---------+------+
```

```
mysql> select * from manytomany_app_course_student;
+----+-----------+------------+
| id | course_id | student_id |
+----+-----------+------------+
|  1 |         1 |          1 |
|  2 |         1 |          3 |
|  3 |         2 |          1 |
|  4 |         2 |          2 |
|  5 |         3 |          2 |
|  6 |         3 |          3 |
+----+-----------+------------+
```

Here,
If we delete any parent record which has the child record, then corresponding child record will not delete in the child table, it will delete the relationaships link from intermediate table.

If we delete all parent records then all relationships will be deleted from the intermediate table, no child record will be deleted.

Field name restrictions when we are creating django model field names.

1. We can't use python keywords as field names but we can use database keywords as field name

For example:

```
delete = models.IntegerField()          ------> right
def = models.IntegerField()             ------> wrong
```

2. We can use only underscore special character in the field name but we can’t use any other special characters

For example:

```
first_num = models.IntegerField()            ------> right
first%num = models.IntegerField()            ------> wrong
```

3. We can use any number of underscores individually in a single field name but we can not use two or more underscores in a sequence.

For example:

```
my_first_emp_id = models.IntegerField()      ------> right
emp___id = models.IntegerField()             ------>  wrong
```

4. We can use digits in the field name but the field name should not startwith digit.

For example:

```
first_num1 = models.IntegerField()           ------> right
1first_num = models.IntegerField()           -------> wrong
```

Note: Some of the field types that we are using regularly in Django models.

```
   models.SmallIntegerField()
   models.IntegerFiled()
   models.BigIntegerField()

   models.PositiveSmallIntegerField()
   models. PositiveIntegerFiled()
   models. PositiveBigIntegerField()

   models.FloatField()
   models.DecimalField(max_digites=13,decimal_places=2)    ex : 30457.60

  models.CharField()
  models.TextField()
  models.EmailField()
```

```
    models.DateField() -- mm/dd/yyyy
    models.TimeField()
    models.DateTimeField()

    models.FileField()
    models.ImageField()


    models.OneToOneField()
    models.ForeignKey()
    models.ManyToManyField()

    models.AutoField()
```

Customizing the Model class Table name and model field Column names.

```
from django.db import models

class Employee(models.Model):
    eno = models.IntegerField(primary_key=True)
    ename = models.CharField(max_length=30, db_column='employee_name')
    email = models.EmailField(unique=True) # admin@gmail.com
    mobile = models.IntegerField(unique=True, null=True, blank=True)
    address = models.TextField(default='Hyderabad')

    class Meta:
        db_table = 'employee'
```

Here, Meta class giving some extra information to the current Model class.
db_table attribute represents the model table name. Here django creates table name is like **"employee".**
By default django creates table names like **applicationName_modelName.**

db_column optional attribute represents required format column name in the database table.

blank=True accepting a blank value for a model field in form presentation.
Null=True accepting a null value for a model field in the database table.