Django Model Relationships:

- Models follow RDBMS like any other database.
- RDBMS stands for Relational Database Management System.
- Oracle, SQL Server, MySQl, Django models.... are the examples of RDBMS.
- In RDBMS, all tables must have Relationships.
- Relationships are used to avoid duplicate data in the tables.
- Database relationships are used to associate records on the basis of a key or id, resulting in improved data maintenance, query performance and less duplicate data, among other things.

Django models supports different types of Relationships:

1. One to One Relationship
2. Many to One Relationship
3. Many to Many Relationship.

4. One to Many Relationship ----->> Not Support

1. One to One Relationship :

- In One-to-One Relationship, One child record can depend on only one parent record.
- If we try to depend multiple child records on single parent record in One-to-One Relationship then It will return Error.
- We use OneToOneField to implement One-to-One Relationship.
- We create the relationaship field in child model by passing the some inputs.

  author = models.OneToOneField(Parent, on_delete = models.CASCADE)

Q. Create a Django Project using OneToOne Model Relationship

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

Step2: Create a Application Name like **OneToOne_App**

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

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

```
DATABASES = {
```

```
    'default': {
        'ENGINE'  :  'django.db.backends.mysql',
        'NAME' : '7am_otodb',
        '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.OneToOneField(Student, on_delete=models.CASCADE)

    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','student_id']
```

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.

mysql> select * from onetoone_app_student;

```
+----+-----+--------+--------------+
| id | sno | sname  | marks |
```

```
+----+-----+--------+-------------+
|  1 | 101 | Virat  |    85       |
|  2 | 102 | Rohit  |    90       |
|  3 | 103 | Surya  |    75       |
|  4 | 104 | Dravid |    95       |
+----+-----+--------+-------------+
```

```
mysql> select * from onetoone_app_course;
+----+-----+--------+------+--------------------+
| id | cno | cname  | cfee | student_id |
+----+-----+--------+------+--------------------+
|  1 | 201 | Python | 5000 |          4 |
|  2 | 202 | Django | 6000 |          3 |
|  3 | 203 | RESTAPI| 4000|          2 |
| 4  | 204 | MySQL |  3000 |          1 |
+----+-----+--------+------+----------------+
```

Now you can Perform all required CURD operations using this admin site presentation.

on_delete:

Django Model provides **on_delete** attribute option with several values. By default it contains CASCADE value for on_delete attribute.

Django providing several cascading values which are automatically converting into MySQL Database related on_delete cascading values.

For example,

Django Models	**Databases (MySQL)**
----------	----------
1. DO_NOTHING	1. No_Action ----->> default
2. SET_NULL	2. Set_null
3. SET(value)	3. Set_default(value)
4. CASCADE -------->> default	4. Cascade

Exaplanation :

In databases, No_Action is the default cascading rule so we can not delete or update the parent records which has the child records.

In Django Models, CASCADE is the default cascading rule so we can delete or update the parent record which has the child records also.

DO_NOTHING

- If we set on_delete = models.DO_NOTHING in the student field of Course model then we can't delete or update the parent records which has child records.
- Goto Course model class and modify the student field like below
- **student = models.OneToOneField(Student , on_delete = models.DO_NOTHING)**

- Now run makemigrations, migrate and runserver commands, Next goto admin site and refresh.
- Select Virat parent record and try to delete, It will throw **IntegrityError**.

SET_NULL:

- If we set on_delete = models.SET_NULL in the student field of Course model then we can delete the parent record but the corresponding child record student_id become NULL value.

student = models.OneToOneField(Student , on_delete = models.SET_NULL, null = True)

SET(value):

- If we set on_delete = models.SET(3) then it will allow to delete the parent records and also it will replace the student_id of the corresponding child with 3.
- That means, we are remapping the child record to another parent.
- Note : If another parents records not available in parent table then it is not possible to remapping the another parent reference into a child record where a parent reference is there.
- Then it returns one error like bellow, IntegrityError at /admin/OTO_App/student/ (1452, 'Cannot add or update a child row: a foreign key constraint fails (`online7pm_oto_db`.`oto_app_course`, CONSTRAINT `OTO_App_course_student_id_597df60d_fk_OTO_App_student_id` FOREIGN KEY (`student_id`) REFERENCES `oto_app_student` (`id`))')

- Note: on_delete = models.SET(int) is not working in OneToOneFiled relationship, because here one child must be depend on only one parent record.
- So when we are remapping the another parent id into child record where parent refference is there then we are getting exception like IntegrityError at /admin/OTO_App/student/

CASCADE

- If we set on_delete = models.CASCADE in the student field of the Course models then we can delete the Parent and also the corresponding child record will also delete.