

التعامل مع قواعد بيانات Django

مفهوم ORM و Migrations

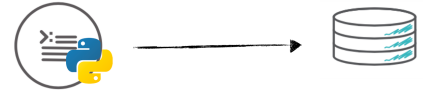
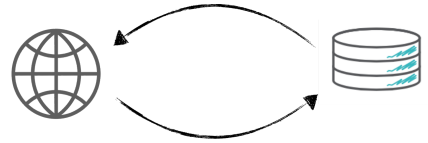
قبل البدء بكتابة Model لابد من التعرف على مفهومين أساسيين وهما:

- مفهوم (Object Relational Mapper(ORM
- مفهوم Migrations

أولاً: مفهوم (Object Relational Mapper(ORM

في معظم تطبيقات الويب نحتاج للتفاعل مع البيانات عن طريق (تخزين، عرض، تعديل، حذف) البيانات. و يقوم Django بتوفير طريقة سهلة للتعامل مع قواعد البيانات عن طريق ORM.

يساعد ORM بتحويل أكواد Python إلى database schema & tables من دون الحاجة لفهم تفاصيل قواعد البيانات وهذا يساعد في تسريع Web Development لأننا نستطيع تعريف قواعد البيانات باستخدام أكواد مكتوبة بلغة Python.



مكونات Object Relational

(Mapper(ORM

أولاً: Model

ويتم تمثيله عن طريق Python Class بحيث يمثل البيانات الخاصة بتطبيق معين، ونقوم بتعريف fields الخاصة بالبيانات ونوعها بداخل هذا Class. مثل: الاسم ورقم الطالب إذا كنا نريد بناء قاعدة بيانات خاصة بالطلاب.



- Student Name
- Student ID

ثانياً: Migrations

يساعد في تحديث قاعدة البيانات المكتوبة بلغة SQL بحيث تكون متوافقة مع Model الذي تم كتابته بلغة Python.

Migrations



مثال توضيحي:

لو كان لدينا Model يحتوي fields خاصة بالطلاب (الاسم ورقم الطالب)، وبعد فترة زمنية أردنا تغيير Model بحيث يشمل المعدل التراكمي للطالب، في هذه الحالة سوف نقوم بإضافة المعدل بداخل Model وأيضا نحتاج لعمل نفس التغييرات على قاعدة البيانات وهنا يأتي دور Migrations التي تقوم بتحديث قواعد البيانات بحيث تكون متطابقة مع ملف Model.

Student ID	First Name	Last Name
623456	Sara	Ahmed
725604	Lama	Hassan

+
GPA
4.7
4.5

نظرة عامة على قواعد بيانات Django

أنواع SQL Database Servers

قاعدة البيانات الافتراضية في Django هي SQLite ولكن يمكننا استخدام قواعد بيانات أخرى (SQL database servers) مثل: PostgreSQL و MySQL و Microsoft SQL Server



عند الدخول على ملف settings.py الموجود بداخل مجلد Movies نلاحظ أن قواعد البيانات الافتراضية هي SQLite3

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    }  
}
```

يوجد أيضا installed apps وتحتوي التطبيقات الافتراضية التي ينشأها Django.

```
32  
33 INSTALLED_APPS = [  
34     'django.contrib.admin',  
35     'django.contrib.auth',  
36     'django.contrib.contenttypes',  
37     'django.contrib.sessions',  
38     'django.contrib.messages',  
39     'django.contrib.staticfiles',  
40     'MoviesListApp',  
41 ]
```

قبل البدء بالتعامل مع البيانات يمكنك استعراض معلومات عامة عن ماتحتويه قواعد بيانات django الافتراضية.

أولا: استعراض migrations عن طريق الأمر:

```
python manage.py showmigrations
```

نلاحظ ظهور جميع التعديلات التي حدثت على database التابعة لكل تطبيق.

```
admin
[X] 0001_initial
[X] 0002_logentry_remove_auto_add
[X] 0003_logentry_add_action_flag_choices
auth
[X] 0001_initial
[X] 0002_alter_permission_name_max_length
[X] 0003_alter_user_email_max_length
[X] 0004_alter_user_username_opts
[X] 0005_alter_user_last_login_null
[X] 0006_require_contenttypes_0002
[X] 0007_alter_validators_add_error_messages
[X] 0008_alter_user_username_max_length
[X] 0009_alter_user_last_name_max_length
[X] 0010_alter_group_name_max_length
[X] 0011_update_proxy_permissions
[X] 0012_alter_user_first_name_max_length
contenttypes
[X] 0001_initial
```

ثانياً: إضافة تعديلات ثم تحديث قواعد البيانات عن طريق الأمر:

```
python manage.py migrate
```

ثالثاً: استعراض الجداول في قواعد البيانات عن طريق الأمر:

```
python manage.py dbshell
```

هذا الأمر يساعد Django في التواصل مع قاعدة البيانات وبالتالي استعراض الجداول بداخلها عن طريق الأمر:

```
.tables
```

نلاحظ ظهور قائمة بجميع الجداول الموجودة لدينا داخل قاعدة البيانات.

عندما نريد استعراض أحد هذه الجداول (على سبيل المثال: `django_migrations`) نقوم بكتابة SQL Query بالشكل التالي:

```
select * from django_migrations;
```

للخروج من terminal نقوم بكتابة:

```
.exit
```

إنشاء Model

لإنشاء Model نقوم باتباع الخطوات التالية:

- فتح ملف `model.py` الموجود داخل مجلد التطبيق `MoviesListApp`
- إضافة الأعمدة (`attributes`) التي نريد وجودها في جدول قاعدة البيانات (على سبيل المثال: اسم الفلم وتاريخ إنطلاقه) مع تحديد نوع الأعمدة (`Fields`) وخصائصها.
- كتابة الأكواد التالية:

```
from django.db import models

class Movies_Info(models.Model):
    name = models.CharField(max_length=100, help_text="The name of the Movie.")
    date = models.DateField(verbose_name="Date the Movie was released.")
```

من الأكواد السابقة نلاحظ:

- يمثل `name` العمود الأول: اسم الفلم وهو من نوع `CharField` ويحمل الخصائص `max_length` و `help_text`.
- يمثل `verbose_name` وصف `Field`.
- يقوم `class Movies_Info` بوراثة `Model class` الخاص بـ `Django` وهي خطوة ضرورية في كل مره نريد عمل `Model`.

- تنفيذ التعديلات التي قمنا بكتابتها بداخل `model.py` على قواعد البيانات عن طريق كتابة:

```
python manage.py makemigrations MoviesListApp
```

الأمر `makemigrations` يعرض التعديلات المتوقع حدوثها على قاعدة البيانات بحيث تكون متطابقة مع `Model` ونلاحظ أنه تم إنشاء `Model Movies_Info` ونتج عن ذلك الملف التالي:

```
MoviesListApp/migrations/0001_initial.py
```

بداخل هذا الملف نلاحظ أنه تم تحديد جداول قاعدة البيانات مع `fields` الذي قمنا بإضافتها. أيضا تم إضافة `field` خاص بـ `ID` بحيث يكون `primary key` يشير إلى `Row` أو `Object`.

- نقوم بإدخال الأمر التالي:

```
python manage.py sqlmigrate MoviesListApp 0001
```

حيث يمثل `0001` بداية اسم ملف `Migrations` << ملف `initial.py_0001` بعد تنفيذ هذا الأمر سوف نلاحظ أن هذا الأمر قام بطباعة `SQL Code` الذي سوف يتم تنفيذه ويشمل عمل إنشاء لجدول `Movies_Info` ويحتوي `fields` التالية: `ID` و `name` و `date`.

- الآن نقوم بتحديث قاعدة البيانات عن طريق تنفيذ `migrations` باستخدام الأمر

```
python manage.py migrate
```

لاستعراض الجدول الذي قمنا بإضافته نقوم بكتابة الأمر

```
python manage.py dbshell
```

ثم نقوم بكتابة `tables`. وهنا سوف يظهر لنا الجدول الجديد (`MoviesListApp_movies_info`)

- يمثل الجزء الأول من اسم الجدول اسم التطبيق (MoviesListApp)
- يمثل الجزء الثاني اسم Model الذي قمنا بإضافته (movies_info)

الفرق بين أوامر Migrations

الفرق بين الأوامر [makemigrations]، [migrate]، [sqlmigrate] هو التالي:

أمر makemigrations

- يساعد الأمر [python manage.py makemigrations App_Name] بإنشاء migration scripts للتطبيقات الموجودة في المشروع.

أمر migrate

- يساعد الأمر [python manage.py migrate] بتطبيق التعديلات على قاعدة البيانات.

أمر sqlmigrate

- يساعد الأمر [python manage.py sqlmigrate App_Name migration_name] بطباعة sql الخاص بملف migration_name .

أنواع Fields المستخدمة في Model

في Django توجد أنواع Field متعددة يمكن استخدامها لتعريف البيانات داخل في Models ومن أشهرها التالي:

النوع	استخدامه
CharField	يستخدم لتخزين جمل قصيرة
TextField	يستخدم لتخزين جمل طويلة
IntegerField	يستخدم لتخزين الأرقام من (-2147483648) إلى (2147483647)
FloatField	يستخدم لتخزين الأرقام من نوع float كما في لغة Python
DateField	يستخدم لتخزين بيانات اليوم ويمثل datetime.date في لغة Python
BooleanField	يستخدم لتخزين القيم True و False
EmailField	يستخدم للتحقق مما إذا كانت الجملة تعبر عن بريد إلكتروني أم لا
URLField	يستخدم للتحقق مما إذا كانت الجملة تعبر عن URL صحيح أم لا
JSONField	يستخدم لتخزين بيانات JSON

وللاطلاع على المزيد من أنواع Field عن طريق الرابط:


<https://docs.djangoproject.com/en/3.2/ref/models/fields/>

تنصيب SQLite Browser

يمكن استخدام SQLite Browser لاستعراض البيانات التي قمنا بتخزينها في قواعد البيانات.

نظام MacOS

<https://satr.codes/courses/c966f82e-9c5f-472c-bb9e-5fabeb2f4a08/session/e70c72c9-88ad-4e7b-a3b7-6cca8dbd54b4/view>

 منصة سطر التعليمية • satr.codes

نظام Windows

<https://satr.codes/courses/c966f82e-9c5f-472c-bb9e-5fabeb2f4a08/session/6220c77b-c923-4691-afb4-72bc0eb00bfb/view>

 منصة سطر التعليمية • satr.codes

عمليات CRUD في قواعد البيانات

أولاً: عمليات CRUD في SQL

عمليات crud operations هي اختصار لكلمة (Create, Read, Update, Delete)

مثال: لنفرض أن لدينا قاعدة بيانات تحتوي جدول **Movie** ويحتوي الأعمدة التالية:

director	publisher	title

أولاً: عملية إضافة البيانات (Create)

```
insert into Movie values ('Movie1', 'x_publisher', 'director 1')
```

title	publisher	director
-------	-----------	----------

Movie1	x_publisher	director1
--------	-------------	-----------

ثانيا: عملية القراءة أو استرجاع البيانات (Read)

```
select * from Movie;
```

ثالثا: عملية تحديث البيانات (Update)

```
update Movie set publisher = 'publisher1' where title='Movie 1';
```

title	publisher	director
Movie1	publisher1	director1

رابعا: عملية حذف البيانات (Delete)

```
delete from Movie where title='Movie1'
```

title	publisher	director

مقارنة بين عمليات CRUD في SQL و Django

بالمقابل يمكن تنفيذ عمليات CRUD في Django

أولا: عملية Create

- **Create In SQL**

Insert into Publisher values (.....);

- **Create In Django**

publisher = Publisher(.....)

publisher.save()

publisher = Publisher.objects.create(.....)

ثانياً: عملية Read

- **Read In SQL**

Select name from Publisher;

- **Read In Django**

Publisher.objects.get(name='Publisher_A')

Publisher.objects.all()

Publisher.objects.filter(name='Publisher_A')

ثالثاً: عملية Update

• Update In SQL

Update Publisher set name = 'Publisher_B' where name = 'Publisher_A';

• Update In Django

`Publisher.objects.filter(name='Publisher_A').update(name='Publisher_B')`

رابعاً: عملية Delete

• Delete In SQL

Delete from Publisher where name='Publisher_A';

• Delete In Django

`Publisher.objects.get(name='Publisher_A').delete()`

إضافة Model Classes

قبل التعرف على عمليات CRUD في Django سوف نضيف المزيد من Models.

بداخل ملف `models.py` سوف ننشأ ثلاثة Models:

- الأول: Publisher
- الثاني: Contributor
- الثالث: Review

```
from django.db import models

class Publisher(models.Model):
```

```

    name = models.CharField(max_length=50, help_text="The name
of the Publisher.")

    website = models.URLField(help_text="The Publisher's websi
te.")

    email = models.EmailField(help_text="The Publisher's email
address.")

class Contributor(models.Model):

    first_name = models.CharField(max_length=50, help_text="Th
e contributor's first name.")

    last_name = models.CharField(max_length=50, help_text="The
contributor's last name or name.")

    email = models.EmailField(help_text="The contact email for
the contributor.")

class Review(models.Model):

    content = models.TextField(help_text="The Review text.")

    rating = models.IntegerField(help_text="The rating the rev
iewer has given.")

    date_created = models.DateTimeField(auto_now_add=True, hel
p_text="The date and time the review was created.")

```

نقوم بتطبيق التعديلات عن طريق الأمر:

```
python manage.py makemigrations MoviesListApp
```

نلاحظ أن الامر makemigrations أنشأ التالي:

- أنشأ ثلاثة Models
 - أنشأ ملف migration داخل مجلد migrations وهو الملف الناتج عن التعديل ويحتوي اسم الجداول و fields
- لاحظ أن الملفات الناتجة تحتوي على رقم تسلسلي فمثلا عند تنفيذ أول migrations سوف يبدأ الملف برقم 001 وثاني migrations سوف يكون 002 وهكذا

لعرض جميع migrations نكتب الأمر

```
python manage.py showmigrations
```

لاحظ أن الملفات التي تحوي بجانبها علامة x تمثل migrations التي تم تطبيقها، ولكن الملف الذي أضفناه ليس بجانبه علامة x وذلك بسبب أننا لم نطبق migrate.

نقوم بتحويل Model Classes إلى Databases Tables عن طريق الأمر

```
python manage.py sqlmigrate MoviesListApp 0001
```

نلاحظ ظهور sql code يقوم بإنشاء Tables التي تمت إضافتهم
ننفذ migrate عن طريق الأمر:

```
python manage.py migrate
```

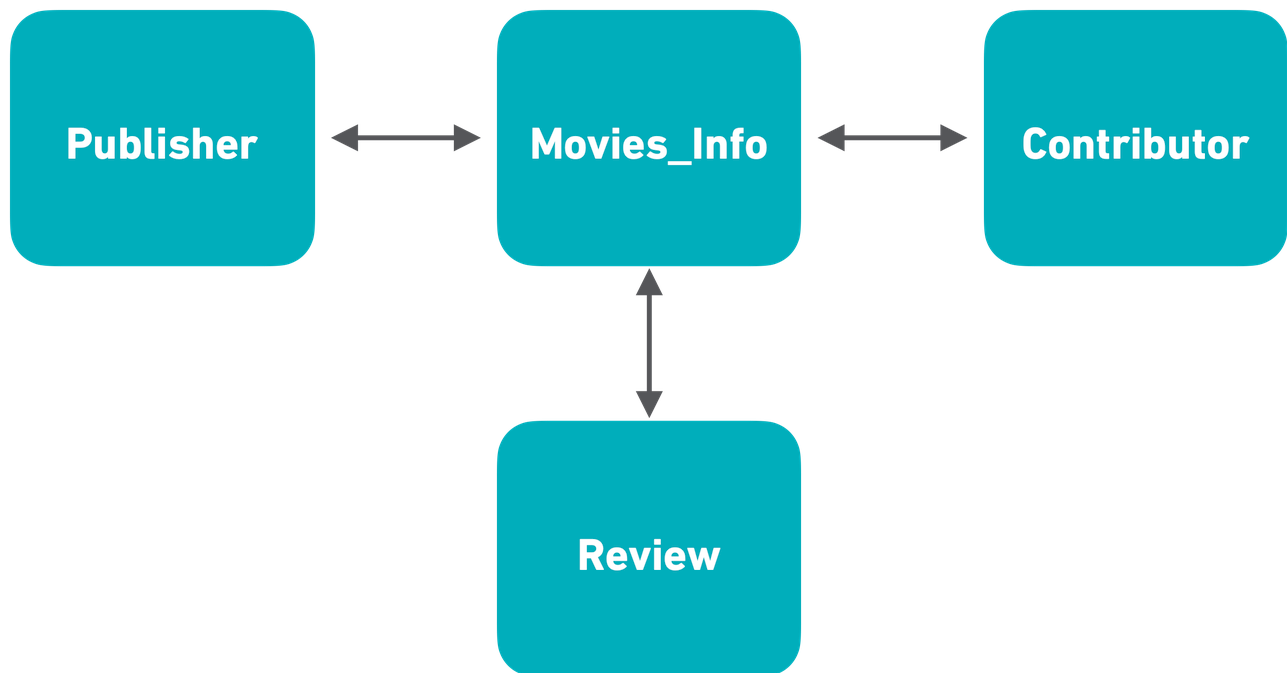
عند الرجوع وكتابة الأمر:

```
python manage.py showmigrations
```

نلاحظ أن ملف migration الذي تمت إضافته تظهر عنده علامة x وهذا يعني أن migrations تم تنفيذها.
تذكر: يمكنك الاطلاع على قاعدة البيانات عن طريق SQLite Browser.

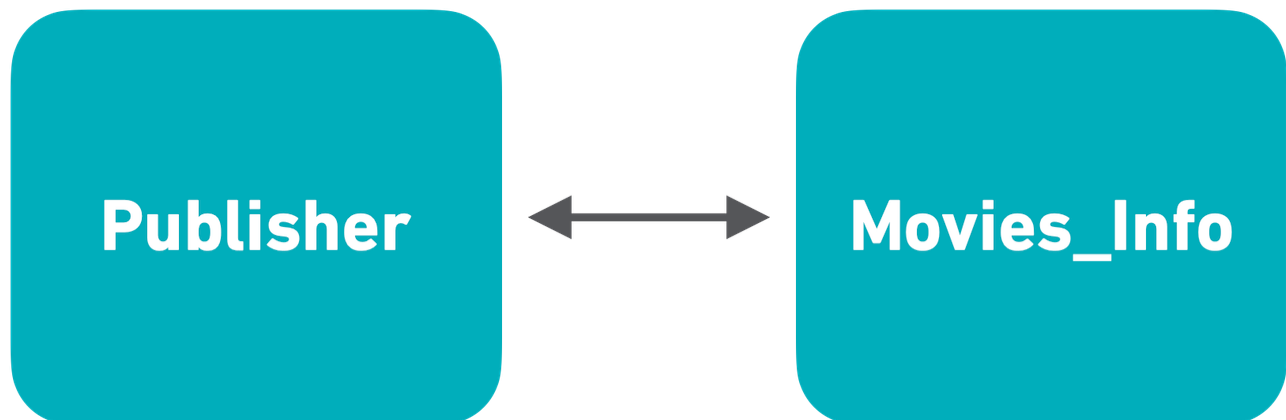
إنشاء العلاقات

الهدف من إنشاء العلاقات بين الجداول في قواعد البيانات هو ضمان عدم وجود تكرار في البيانات بين الجداول والحفاظ على تناسق البيانات.



يوجد لدينا الآن مجموعة من Model Classes وهم Publisher و Movies_Info و Contributor و Review
كل Model يمكن أن تراه كجدول في قاعدة بيانات، لذا نستطيع إنشاء علاقات بين هذه Models كالتالي:

أولاً: علاقة Many to One



تعتبر العلاقة بين Publisher و Movies_Info هي علاقة من نوع Many to One بحيث أن أي publisher لديه أكثر من فلم، ويمكن إنشاء العلاقة عن طريق كتابة التالي بداخل Movies_Info model:

```
publisher = models.ForeignKey(Publisher, on_delete=models.CASCADE)
```

نستخدم on_delete لتحديد الاجراء الذي نتبعه عند حذف object حيث يوجد لدينا عدة خيارات وهي:

- الأول : CASCADE عند حذف Publisher فسوف يتم حذف Movies التابعة له
- الثاني : PROTECT هذا النوع يمنع حذف Publisher إلا في حال حذف جميع Movies المتعلقة ب Publisher
- الثالث: SET_NULL عند حذف Publisher سوف تكون قيمة Publisher في جدول Movies_Info تساوي null
- الرابع: SET_DEFAULT عند حذف Publisher سوف تكون قيمة Publisher في جدول Movies_Info تساوي القيمة الافتراضية التي اخترناها

ملاحظة:

في حال ظهور هذه الرسالة "Unresolved reference publisher" قم بنقل Movies_Info أسفل Publisher

أيضا توجد علاقة من نوع Many to One:

- بين Review و Movies_Info لأن الفلم الواحد توجد عليه عدة تعليقات.
- بين Review و User لأن الشخص الواحد يستطيع كتابة عدة تعليقات.

ويمكن إنشاء العلاقة عن طريق كتابة التالي بداخل Review model:

```
from django.contrib import auth
```

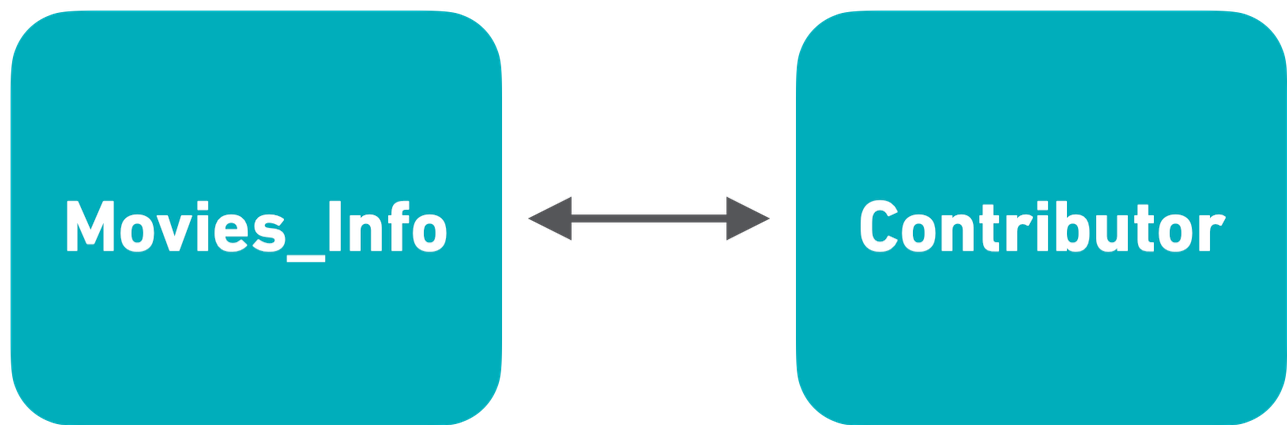
```

creator = models.ForeignKey(auth.get_user_model(), on_delete=models.CASCADE)

movie = models.ForeignKey(Movies_Info, on_delete=models.CASCADE,
                           help_text="The Movie that this review is for.")

```

ثانياً: علاقة Many to Many



العلاقة بين Contributor و Movies_Info تعتبر من نوع Many to Many لأن الفلم الواحد ممكن يكون فيه عدة مساهمين Contributors (مثلاً: ممثلين ومخرجين) وأيضا One Contributor قد يكون مشارك بأكثر من فلم، ويمكن إنشاء العلاقة عن طريق كتابة التالي بداخل model :Movies_Info:

```

contributors = models.ManyToManyField('Contributor', through="MovieContributor")

class MovieContributor(models.Model):
    class ContributionRole(models.TextChoices):
        ACTOR = "ACTOR", "Actor"
        DIRECTOR = "DIRECTOR", "Director"
    movie = models.ForeignKey(Movies_Info, on_delete=models.CASCADE)

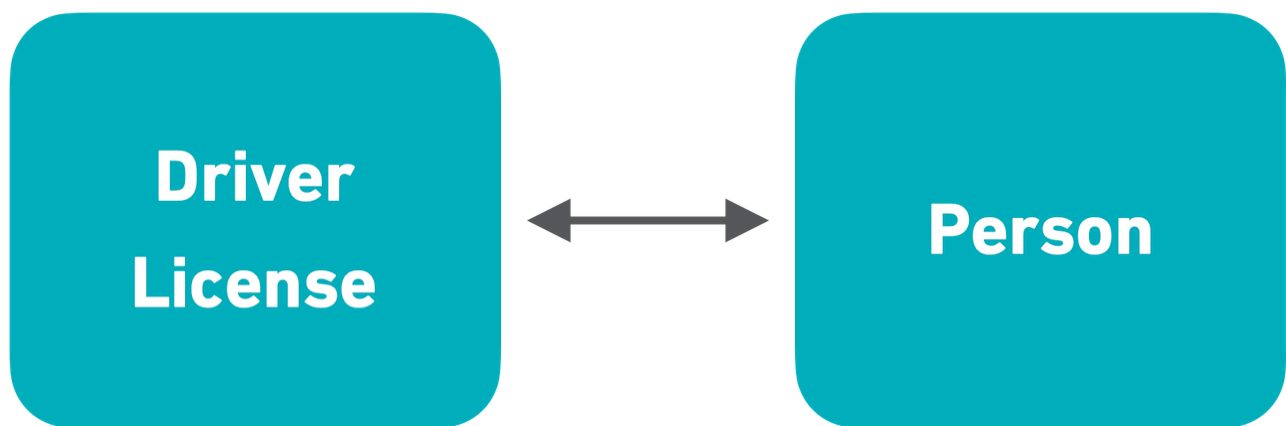
```

```
contributor = models.ForeignKey(Contributor, on_delete=models.CASCADE)

role = models.CharField(verbose_name="The role this contributor had in the movie.", choices=ContributionRole.choices, max_length=20)
```

لكتابة العلاقة من نوع Many to Many لابد من كتابة through و نستخدمها إذا كان لدينا معلومات إضافية عن العلاقة.

ثالثاً: علاقة One to One



يستخدم هذا النوع من العلاقات بشكل قليل، و مثال على هذا النوع: العلاقة بين driver license و Person لأن الشخص ممكن يحصل على رخصة قيادة واحدة و هذي الرخصة لا يمكن أن تنتمي إلا لشخص واحد فقط. ويمكن إنشاء العلاقة عن طريق كتابة التالي:

```
person = models.OneToOneField(Person, on_delete=models.CASCADE)
```

ثانياً: عمليات CRUD في Django

تعرفنا بالسابق على CRUD Operations في SQL، لكن استخدام SQL Queries سوف يكون معقد بشكل أكبر عندما يكون لدينا العديد من الجداول، لذلك نحتاج لاستخدام Django's CRUD Operations. لذا Django قامت بتوفير ORM والذي يساعد في تنفيذ عمليات CRUD دون الحاجة لكتابة SQL Queries.

• عملية Create

يمكن إضافة بيانات باستخدام أكثر من طريقة مثل: save() Method و create() Method

أولاً: save() Method

عندما نريد إضافة Publisher Record نقوم بالتالي:

- كتابة الأمر `python manage.py shell`
- تعريف Object ثم مناداة `save()` Method

```
from MoviesListApp.models import Publisher
```

```
publisher = Publisher(name='Publisher_A', website='https://www.Publisher_A.com', email='Publisher_A@gmail.com')
publisher.save()
```

- استرجاع قيمة `email`

```
publisher.email
```

- تعديل قيمة `email`

```
publisher.email = 'Publisher_A2@gmail.com'
publisher.save()
```

ثانياً: `create()` Method

يمكن إضافة البيانات بخطوة واحدة فقط عن طريق إرسال القيم بداخل `.create()` Method

```
from MoviesListApp.models import Contributor
```

```
contributor = Contributor.objects.create(first_name="Ali", last_name="Ahmed", email="Ali@example.com")
```

- عملية `Read`

يمكن استرجاع البيانات أو قرائتها باستخدام أكثر من طريقة مثل: `get()` ، `all()` ، `filter()` methods

أولاً: `get()` Method

يمكن استرجاع البيانات عن طريق `get()` Method بالشكل التالي:

```
from MoviesListApp.models import Publisher
```

```
publisher = Publisher.objects.get(name='Publisher_A')
publisher
publisher.name
publisher.website
```



```
publisher.email
```

نلاحظ أن Method `get()` ترجع `One Object` و هذا يعتبر من السليبيات في هذه Method، فمثلا لو كان لدينا أكثر من `Publisher` بنفس الاسم سوف يظهر لنا `error message`، أيضا لو كنا نبحث عن اسم غير موجود سوف يظهر لنا `error message`.

ثانيا: `all()` Method

يمكن استرجاع البيانات عن طريق `all()` Method بالشكل التالي:

```
from MoviesListApp.models import Contributor

Contributor.objects.all()

contributors = Contributor.objects.all()

contributors[0]

contributors[0].last_name
```

نلاحظ أن `all()` Method تقوم بإرجاع مجموعة من `Objects` بدلا من `Object` واحد. ويمكن إرجاع `Object` واحد عن طريق `index` كما في المثال السابق `[contributors[0]]`.

ثالثا: `filter()` Method

يمكن استرجاع البيانات عن طريق `filter()` Method بالشكل التالي:

```
from MoviesListApp.models import Contributor

Contributor.objects.create(first_name="Asma", last_name="Ahmed", email="Asma@example.com")
Contributor.objects.create(first_name="Asma", last_name="Saleem", email="Rana@example.com")

Contributor.objects.filter(first_name='Asma')
contributors = Contributor.objects.filter(first_name='Asma')
contributors[0].last_name
contributors[1].last_name
Contributor.objects.filter(first_name='Nobody')
```

نلاحظ أن `filter()` Method تشابه `all()` Method ولكن بدلا من إرجاع كل `Objects` نستطيع وضع شرط معين، مثلا: `'first_name='Asma'`

• عملية Update

يمكن تحديث البيانات أو تعديلها باستخدام `update() method` بالشكل التالي:

```
from MoviesListApp.models import Contributor
Contributor.objects.get(last_name='Salem').first_name
Contributor.objects.filter(last_name='Salem').update(first_name='Rana')
Contributor.objects.get(last_name='Salem').first_name
```

• عملية Delete

يمكن حذف البيانات باستخدام `delete() method` بالشكل التالي:

```
from MoviesListApp.models import Contributor
Contributor.objects.get(last_name='Salem').delete()
Contributor.objects.get(last_name='Salem')
```


نلاحظ ظهور *error message* يفيد بعدم وجود *Object* وهو

MoviesListApp.models.Contributor.DoesNotExist : Contributor matching query does not exist

استخدام قواعد بيانات PostgreSQL

تنصيب برنامج PostgreSQL - نظام MacOS

<https://satr.codes/courses/a7444c95-e4fb-46c0-9d77-72e68605d5ea/session/e6c7ca91-8a64-4068-9b2c-b745b741a06b/view>

 منصة سطر التعليمية • satr.codes

تنصيب برنامج pgAdmin - نظام MacOS

<https://satr.codes/courses/a7444c95-e4fb-46c0-9d77-72e68605d5ea/session/b6b8430-0-d3e3-491c-9928-b1d9ba187159/view>

 منصة سطر التعليمية • satr.codes

تثبيت برنامج PostgreSQL - نظام Windows

<https://satr.codes/courses/a7444c95-e4fb-46c0-9d77-72e68605d5ea/session/dc00b1f1-f7d4-4e03-95a7-e216fa4a5bb6/view>



منصة سطر التعليمية • satr.codes