EXERCISES QUE TRABAJAREMOS EN EL CUE

- **EXERCISE 1: RELACIONES UNO A MUCHOS**
- **EXERCISE 2: RELACIONES MUCHOS A MUCHOS**
- **EXERCISE 3: RELACIONES UNO A UNO**

EXERCISE 1: RELACIONES UNO A MUCHOS

Para este tipo de relación, el registro de una tabla se puede asociar con uno o más registros de otra tabla.

Vamos a suponer que:

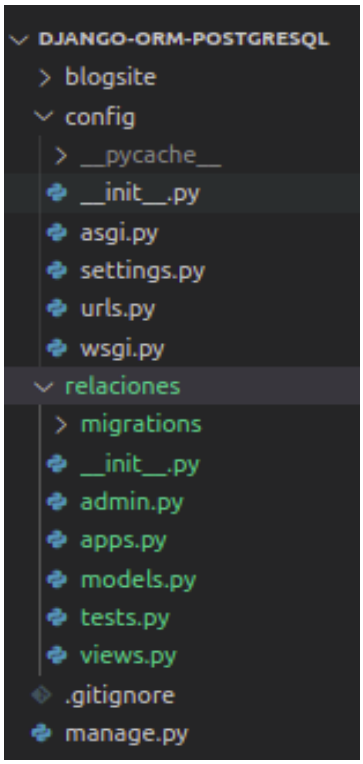
- La compañía de automóviles puede tener uno o más modelos de automóviles.
- El modelo de automóvil sólo puede pertenecer a una compañía de automóviles.

ModeloCarro	Compañia
Cronos	Fiat
Argo	Fiat
Pulse	Fiat
Aveo	Chevrolet
Onix Turbo RS	Chevrolet
Camaro	Chevrolet

Crearemos una nueva aplicación llamada relaciones de nuestro proyecto: `django_orm_postgres`. Procedemos a crear nuestra aplicación relaciones:

```
(django-orm-postgresql)$ django-admin startapp relaciones
```

La estructura del proyecto es:



Registramos la aplicación en el `settings.py` del proyecto que se encuentra dentro de la carpeta `config`.

CONFIG/SETTINGS.PY

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blogsite.apps.BlogsiteConfig',
    'relaciones.apps.RelacionesConfig',
]
```

Seguidamente, creamos los modelos:

RELACIONES/MODELS.PY

```python
from django.db import models

class Automotriz(models.Model):
    nombre = models.CharField(max_length=255)

    def __str__(self):
        return self.nombre

class ModeloCarro(models.Model):
    nombre = models.CharField(max_length=255)
    automotriz = models.ForeignKey(Automotriz,
on_delete=models.SET_NULL, blank=True, null=True)

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

Procedemos a realizar las migraciones correspondientes:

```
$ python manage.py makemigrations

$ python manage.py migrate
```

Observamos las relaciones que el modelo ORM de Django ha generado automáticamente, partiendo de los modelos y sus relaciones:

```
project_orm_django=# \d relaciones_modelocarro
                                  Table "public.relaciones_modelocarro"
    Column     |          Type          | Collation | Nullable |             Default
---------------+------------------------+-----------+----------+---------------------------------
 id            | bigint                 |           | not null | generated by default as identity
 nombre        | character varying(255) |           | not null |
 automotriz_id | bigint                 |           |          |
Indexes:
    "relaciones_modelocarro_pkey" PRIMARY KEY, btree (id)
    "relaciones_modelocarro_automotriz_id_b9dac5d0" btree (automotriz_id)
Foreign-key constraints:
    "relaciones_modelocar_automotriz_id_b9dac5d0_fk_relacione" FOREIGN KEY (automotriz_id) REFERENCES relaciones_automotriz(id)
 DEFERRABLE INITIALLY DEFERRED
```

```
project_orm_django=# \d relaciones_automotriz
                                 Table "public.relaciones_automotriz"
 Column |          Type          | Collation | Nullable |             Default
--------+------------------------+-----------+----------+---------------------------------
 id     | bigint                 |           | not null | generated by default as identity
 nombre | character varying(255) |           | not null |
Indexes:
    "relaciones_automotriz_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "relaciones_modelocarro" CONSTRAINT "relaciones_modelocar_automotriz_id_b9dac5d0_fk_relacione" FOREIGN KEY (automotri
z_id) REFERENCES relaciones_automotriz(id) DEFERRABLE INITIALLY DEFERRED
```

Seguidamente, procedemos a realizar manipulaciones en la consola o Shell de Python:

```
(django-orm-postgresql)$ python manage.py shell
Python 3.8.3 (default, Jul  2 2020, 16:21:59)
```

```
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

Importamos los modelos:

```
>>> from relaciones.models import Automotriz, ModeloCarro
```

Creamos los objetos **automotriz**:

```
>>> fiat = Automotriz.objects.create(nombre='Fiat')
>>> chevrolet = Automotriz.objects.create(nombre='chevrolet')
```

Listamos los objetos creados:

```
>>> Automotriz.objects.all()
>>> Automotriz.objects.all().values
```

En la Shell psql de PostgreSQL consultamos los objetos creados:

```
project_orm_django=# SELECT * FROM relaciones_automotriz;
 id |   nombre
----+-----------
  1 | Fiat
  2 | chevrolet
(2 rows)
```

Creamos un objeto de modelos de carros, un modelo de carro **Cronos** de la Automotriz **Fiat**:

```
>>> cronos = ModeloCarro.objects.create(nombre='cronos',
automotriz=fiat)

>>> cronos.automotriz
```

Seguidamente, procedemos a crear un segundo modelo **Argo** de la Automotriz **Fiat**.

```
>>> argo = ModeloCarro.objects.create(nombre='argo')
```

```
>>> argo.automotriz = fiat
>>> argo.save()
>>> argo.automotriz.nombre
```

Django, de manera predeterminada, le brinda un nombre relacionado predeterminado que es `ModelName` (en minúsculas), seguido de `"_set"`. En este caso, sería `modelocarro_set`.

Creamos dos nuevos modelo de carro `aveo` y `camaro` de la automotriz `Chevrolet`.

```
>>> aveo = ModeloCarro.objects.create(nombre='Aveo')
>>> camaro = ModeloCarro.objects.create(nombre='Camaro')

>>> ModeloCarro.objects.all()

>>> chevrolet.modelocarro_set.add(aveo, camaro)
>>> chevrolet.modelocarro_set.all()
<QuerySet [<ModeloCarro: Aveo>, <ModeloCarro: Camaro>]>
```

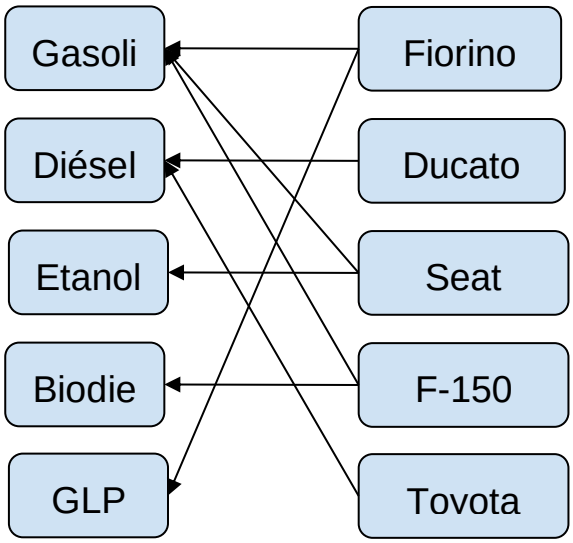
Verificamos en la terminal de psql de PostgreSQL.

```
project_orm_django=# SELECT * FROM relaciones_modelocarro;
 id | nombre | automotriz_id
----+--------+---------------
  1 | cronos |             1
  2 | argo   |             1
  3 | Aveo   |             2
  4 | Camaro |             2
(4 rows)
```

EXERCISE 2: RELACIONES MUCHOS A MUCHOS

Múltiples registros en una tabla están asociados con múltiples registros en otra tabla. Para ejemplificar esta relación de muchos a muchos (`ManyToMany`), supongamos lo siguiente:

- Muchos modelos de automóviles funcionan con el mismo tipo de combustible.
- Un modelo de automóvil puede funcionar con distintos tipos de combustible.



Procedemos a crear el modelo:

RELACIONES/MODELS.PY

```python
from django.db import models

class Automotriz(models.Model):
    nombre = models.CharField(max_length=255)

    def __str__(self):
        return self.nombre

class TipoCombustible(models.Model):
    nombre = models.CharField(max_length=255)

    def __str__(self):
        return self.nombre


class ModeloCarro(models.Model):
    nombre = models.CharField(max_length=255)
    automotriz = models.ForeignKey(Automotriz,
on_delete=models.SET_NULL, blank=True, null=True)
    tipo_combustible = models.ManyToManyField(TipoCombustible)

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

Y a realizar las migraciones:

```
$ python manage.py makemigrations

$ python manage.py showmigrations

$ python manage.py migrate
```

Ingresamos a la Shell de Python:

```
$ python manage.py shell

$ python manage.py shell
Python 3.8.3 (default, Jul  2 2020, 16:21:59)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

Agregamos un modelo de carro:

```
>>> from relaciones.models import Automotriz, ModeloCarro,
TipoCombustible
>>> ducato = ModeloCarro.objects.create(nombre='Ducato')
```

Creamos los tipos de combustible:

```
>>> gasolina = TipoCombustible.objects.create(nombre='Gasolina')
>>> gas = TipoCombustible.objects.create(nombre='Gas')
>>> diesel = TipoCombustible.objects.create(nombre='Diesel')
```

Verificamos los objetos de tipo de **combustible** creados y de modelos de carros:

```
>>> TipoCombustible.objects.all().values
>>> >>> ModeloCarro.objects.all().values
```

Asignamos el tipo de combustible de **Gasolina** al modelo de carro **Ducato**:

```
>>> ducato.tipo_combustible.add(gasolina)
```

Creamos el modelo de carro **F-150**:

```
>>> f150 = ModeloCarro.objects.create(nombre='F-150')
```

Asignamos el tipo de combustible de **Diesel**, **Gasolin** y **Gas** al modelo de carro **F-150**:

```
>>> f150.tipo_combustible.add(diesel)
>>> f150.tipo_combustible.add(gasolina)
>>> f150.tipo_combustible.add(gas)
```

Verificamos los tipos de **combustibles** asignado al modelo de carro **F-150** y **Ducato**:

```
>>> f150.tipo_combustible.all()
>>> ducato.tipo_combustible.all()
```

Procedemos a crear un nuevo modelo de carro **Fiorino**, y un nuevo tipo de **Combustible Biodiesel**, y agregamos el combustible de **gasolina**:

```
>>> fiorino =fiorino.tipo_combustible.create(nombre='Biodiesel')
>>> fiorino.tipo_combustible.create(nombre='Biodiesel')
>>> fiorino.tipo_combustible.add(gasolina)
>>> fiorino.tipo_combustible.all()
```

Visualizamos los tipos de combustibles creados:

```
>>> TipoCombustible.objects.all().values
```

Verificamos los objetos creados en la base de datos consultando sus tablas y relaciones.

Tabla de los modelos de carros creados:

```
project_orm_django=# SELECT * FROM relaciones_modelocarro;
 id | nombre  | automotriz_id
----+---------+---------------
  1 | cronos  |             1
  2 | argo    |             1
  3 | Aveo    |             2
  4 | Camaro  |             2
  5 | Ducato  |
  6 | F-150   |
  7 | Fiorino |
(7 rows)
```

Tabla de los tipos de combustibles creados:

```
project_orm_django=# SELECT * FROM relaciones_tipocombustible;
 id |   nombre
----+-----------
  1 | Gasolina
  2 | Gas
  3 | Diesel
  4 | Biodiesel
(4 rows)
```

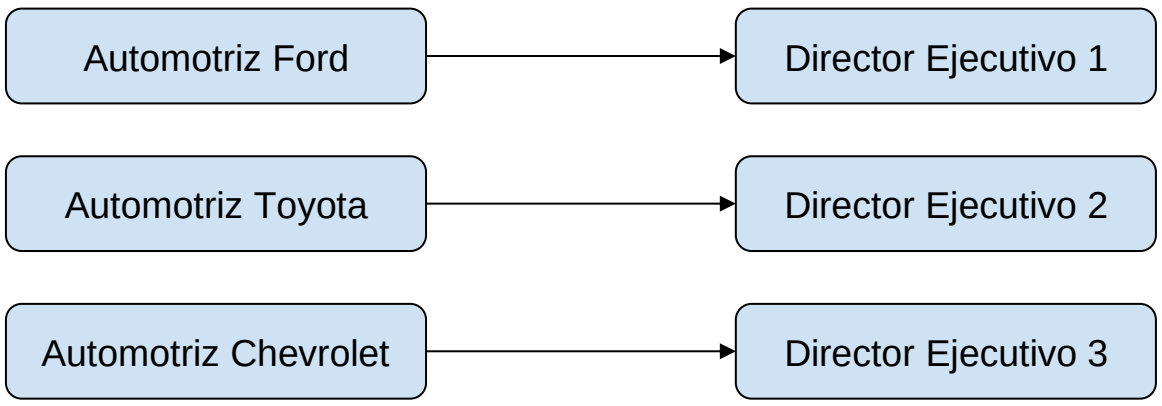
Tabla de las relaciones de los modelos de carros y tipos de combustibles:

```
project_orm_django=# SELECT * FROM
relaciones_modelocarro_tipo_combustible;
 id | modelocarro_id | tipocombustible_id
----+----------------+--------------------
  1 |              5 |                  1
  2 |              5 |                  2
  3 |              6 |                  3
  4 |              6 |                  1
  5 |              6 |                  2
  6 |              7 |                  4
  7 |              7 |                  1
(7 rows)
```

EXERCISE 3: RELACIONES UNO A UNO

Un registro en una tabla está asociado con un solo registro en otra tabla. Suponemos que:

- Una empresa automotriz contiene un sólo Director Ejecutivo.
- El Director Ejecutivo trabaja en una sola empresa automotriz.



Creando el modelo:

RELACIONES/MODELS.PY

```
from django.db import models

class Automotriz(models.Model):
    nombre = models.CharField(max_length=255)

    def __str__(self):
        return self.nombre

class DirectorEjecutivo(models.Model):
    automotriz = models.OneToOneField(Automotriz,
on_delete=models.CASCADE)
    nombre = models.CharField(max_length=255, blank=True)

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

Procedemos a realizar las migraciones:

```
$ python manage.py makemigrations

$ python manage.py showmigrations

$ python manage.py migrate
```

Ingresamos a la Shell de Python:

```
$ python manage.py shell

$ python manage.py shell
Python 3.8.3 (default, Jul  2 2020, 16:21:59)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

Importamos los modelos:

```
>>> from relaciones.models import Automotriz, DirectorEjecutivo
```

Agregando objetos automotriz al modelo:

```
>>> Ford = Automotriz.objects.create(nombre ='Ford')
>>> Toyota = Automotriz.objects.create(nombre ='Toyota')
>>> Chevrolet = Automotriz.objects.create(nombre ='Chevrolet')
```

Consultando los registros ingresados:

```
>>> Automotriz.objects.all().values

<bound method QuerySet.values of <QuerySet [<Automotriz: Ford>,
<Automotriz: Toyota>, <Automotriz: Chevrolet>]>>
```

Agregando los directores ejecutivos a las automotrices:

```
director_toyota = DirectorEjecutivo.objects.create(nombre='Director
Toyota', automotriz=Toyota)
>>> director_ford = DirectorEjecutivo.objects.create(nombre='Director
Ford', automotriz=Ford)
>>> director_chevrolet =
DirectorEjecutivo.objects.create(nombre='Director Chevrolet',
automotriz=Chevrolet)
```

Consultando las instancias:

```
>>> Toyota.directorejecutivo
<DirectorEjecutivo: Director Toyota>

>>> Toyota.directorejecutivo.nombre
'Director Toyota'

>>> Ford.directorejecutivo
<DirectorEjecutivo: Director Ford>

>>> Ford.directorejecutivo.nombre
'Director Ford'

>>> Ch.directorejecutivo.nombre
Chevrolet           ChildProcessError(

>>> Chevrolet.directorejecutivo
<DirectorEjecutivo: Director Chevrolet>

>>> Chevrolet.directorejecutivo.nombre
'Director Chevrolet'
```

Verificamos los objetos creados en la base de datos, consultando sus tablas y relaciones.

Tabla de los automotrices creados:

```
project_orm_django=# SELECT * FROM relaciones_automotriz;
 id |   nombre
----+-----------
  1 | Ford
  2 | Toyota
  3 | Chevrolet
(3 rows)
```

Tabla de las relaciones con el Director Ejecutivo:

```
project_orm_django=# SELECT * FROM relaciones_directorejecutivo;
 id |       nombre        | automotriz_id
----+---------------------+---------------
  1 | Director Toyota     |             2
  3 | Director Ford       |             1
  4 | Director Chevrolet  |             3
(3 rows)
 10 | Chevrolet
(3 rows)
```

Eliminación de registros:

```
Toyota.delete()
(2, {'relaciones.DirectorEjecutivo': 1, 'relaciones.Automotriz': 1})
>>>
```

Consultando los registros:

```
>>> Automotriz.objects.all().values
<bound method QuerySet.values of <QuerySet [<Automotriz: Ford>,
<Automotriz: Chevrolet>]>>

>>> DirectorEjecutivo.objects.all().values
<bound method QuerySet.values of <QuerySet [<DirectorEjecutivo: Director
Ford>, <DirectorEjecutivo: Director Chevrolet>]>>
```

.