EXERCISES QUE TRABAJAREMOS EN EL CUE

- **EXERCISE 1: LOS PROBLEMAS QUE RESUELVEN LAS MIGRACIONES**
- **EXERCISE 2: AGREGANDO UNA APLICACIÓN AL PROYECTO**
- **EXERCISE 3: MIGRACIÓN INICIAL**
- **EXERCISE 4: EL COMANDO MIGRATE**
- **EXERCISE 5: MIGRACIONES POSTERIORES**

EXERCISE 1: LOS PROBLEMAS QUE RESUELVEN LAS MIGRACIONES

Desde la versión 1.7, Django viene con un soporte incorporado para migraciones de bases de datos. Éstas suelen ir de la mano con los modelos: cada vez que se codifica un nuevo modelo, también generas una migración para crear la tabla necesaria en la base de datos.

Django está diseñado para trabajar con una base de datos relacional, almacenada en un sistema de administración de bases de datos relacionales como PostgreSQL, MySQL o SQLite .

En una base de datos relacional, los datos se organizan en tablas. Una tabla de base de datos tiene un cierto número de columnas, pero puede tener cualquier número de filas. Cada columna tiene un tipo de datos específico, como una cadena de cierta longitud máxima, o un entero positivo. La descripción de todas las tablas, con sus columnas y sus respectivos tipos de datos, se denomina esquema de base de datos.

Todos los sistemas de bases de datos compatibles con Django utilizan el lenguaje SQL para crear, leer, actualizar, y eliminar datos en una base de datos relacional. SQL también se usa para crear, cambiar y eliminar las propias tablas de la base de datos.

Trabajar directamente con SQL puede ser bastante engorroso, por lo que Django viene con un mapeador relacional de objetos u ORM para abreviar. Éste asigna la base de datos relacional al mundo de la programación orientada a objetos. En lugar de definir tablas de base de datos en SQL, escribe modelos de Django en Python, los cuales definen campos de base de datos, que corresponden a las columnas en sus tablas de base de datos.

En Django, las migraciones se escriben principalmente en Python, por lo que no requiere conocimientos de SQL, al menos que tenga casos de uso realmente avanzados.

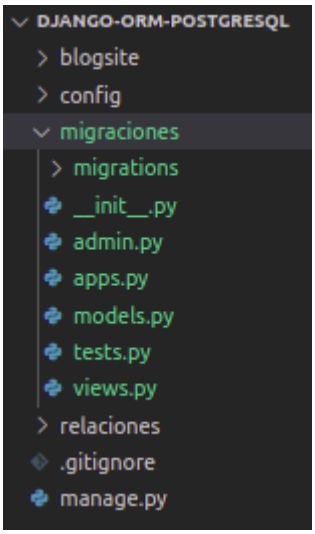
Crear un modelo y luego escribir SQL para crear las tablas de la base de datos sería repetitivo. Las migraciones se generan a partir de sus modelos, asegurándose de que no se repita.

EXERCISE 2: AGREGANDO UNA APLICACIÓN AL PROYECTO

Procedemos a crear nuestra aplicación, que para efectos de esta práctica se llamará migraciones:

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

Esto crea una aplicación del proyecto simple llamada migraciones, que contiene la siguiente estructura de directorios:



Procedemos a crear un modelo que registre el histórico de precios de un vehículo:

MIGRACIONES/MODELS.PY

```
from django.db import models

# Create your models here.

class PrecioHistoricoVehiculos(models.Model):
    fecha = models.DateTimeField(auto_now_add=True)
    modelo = models.CharField(max_length=120)
    precio = models.DecimalField(null=False, decimal_places=2,
max_digits=10)
```

Registramos la aplicación migraciones al proyecto:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # App Local
    'blogsite.apps.BlogsiteConfig',
    'relaciones.apps.RelacionesConfig',
    'migraciones.apps.MigracionesConfig',

]
```

Antes de realizar migraciones, revisemos el modelo en busca de errores usando el comando "**check**". Éste verifica todos los archivos dentro del proyecto Django en busca de errores comunes, y usando el atributo **-tag** podemos especificar qué categoría se revisará.

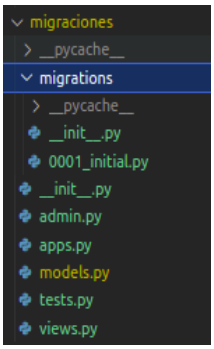
```
$ python manage.py check --tag models

System check identified no issues (0 silenced).
```

EXERCISE 3: MIGRACIÓN INICIAL

Veamos cómo hacer la primera migración para que los cambios realizados en el archivo del modelo se reflejen en la base de datos. En este caso, la base de datos será una PostgreSQL que se adecuó al inicio del proyecto.

```
$ python manage.py makemigrations migraciones 0001
Migrations for 'migraciones':
  migraciones/migrations/0001_initial.py
    - Create model PrecioHistoricoVehiculos
```

Verificamos la estructura de la carpetas del proyecto para identificar el archivo de migraciones recién creado.



Observamos que se ha creado un nuevo archivo de migración dentro de la carpeta de migraciones, llamado `0001_inicial.py`. Este se utilizará para crear la estructura de la tabla inicial en la base de datos que hemos creado y configurado en el proyecto de Django.

Habría notado que en el archivo de migración hay un campo adicional llamado **`"id"`**, el cual se ha definido. Se crea automáticamente como el campo de clave principal en el archivo de migración, pues no hemos definido una clave principal en el archivo `model.py`. Un campo de clave principal se puede definir simplemente agregando el parámetro (**`primary_key=True`**) al campo deseado.

MIGRACIONES/MIGRATIONS/0001_INICIAL.PY

```
# Generated by Django 4.1.1 on 2022-09-19 18:46

from django.db import migrations, models


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='PrecioHistoricoVehiculos',
            fields=[
                ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
                ('fecha', models.DateTimeField(auto_now_add=True)),
                ('modelo', models.CharField(max_length=120)),
                ('precio', models.DecimalField(decimal_places=2,
max_digits=10)),
            ],
        ),
    ]
```

Hemos usado el comando **`makemigrations`** para crear migraciones; sin embargo, los cambios no se reflejarán en la base de datos hasta que apliquemos las migraciones usando el comando **`migrate`**.

EXERCISE 4: EL COMANDO MIGRATE

Antes de aplicar las migraciones, exploremos la base de datos de PostgreSQL para comprobar si se han creado tablas. Usaremos el comando de administración Django `dbshell` para acceder a la base de datos.

```
(django-orm-postgresql) $ $ python manage.py dbshell

project_orm_django=> \d
```

Cuando buscamos tablas con el comando `"\d"`, nos devuelve las creadas previamente, pero aún no se ha creado la tabla `migraciones_preciohistoricovehiculos`.

El comando `sqlmigrate` se puede usar para identificar las consultas SQL que se escribirán en la base de datos sin aplicar los cambios. Éste requiere dos argumentos obligatorios: el nombre de la aplicación y el de la migración.

```
(django-orm-postgresql)$ python manage.py sqlmigrate migraciones
0001_initialBEGIN;
--
-- Create model PrecioHistoricoVehiculos
--
CREATE TABLE "migraciones_preciohistoricovehiculos" ("id" bigint NOT
NULL PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY, "fecha" timestamp
with time zone NOT NULL, "modelo" varchar(120) NOT NULL, "precio"
numeric(10, 2) NOT NULL);
COMMIT;
```

Apliquemos ahora las migraciones usando el comando de `migrate`. Cuando se llama al comando de migración, las aplicará tanto de los archivos de migración definidos por el usuario como de las migraciones del sistema.

```
(django-orm-postgresql)$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, blogsite, contenttypes,
migraciones, relaciones, sessions
Running migrations:
  Applying migraciones.0001_initial... OK
```

Después de una operación de migración exitosa, miraremos la base de datos `postgresql` para verificar si las migraciones se aplicaron correctamente.

```
(django-orm-postgresql)$ python manage.py dbshell

project_orm_django=> \d
******
public   | migraciones_preciohistoricovehiculos         | tabla      |
userdjango
public   | migraciones_preciohistoricovehiculos_id_seq | secuencia |
userdjango
******
```

Observando el esquema creado:

```
project_orm_django=# \d migraciones_preciohistoricovehiculos
                    Table "public.migraciones_preciohistoricovehiculos"
 Column |           Type           | Collation | Nullable |             Default
--------+--------------------------+-----------+----------+---------------------------------
 id     | bigint                   |           | not null | generated by default as identity
 fecha  | timestamp with time zone |           | not null |
 modelo | character varying(120)   |           | not null |
 precio | numeric(10,2)            |           | not null |
Indexes:
    "migraciones_preciohistoricovehiculos_pkey" PRIMARY KEY, btree (id)
```

Las migraciones sólo se aplicarán si se encuentran cambios en los modelos. Cuando no hay migraciones para aplicar, los comandos **`makemigrations`** o **`migrate`** no actuarán.

EXERCISE 5: MIGRACIONES POSTERIORES

MIGRACIONES/MODELS.PY

```python
from django.db import models

# Create your models here.

class PrecioHistoricoVehiculos(models.Model):
    fecha = models.DateTimeField(auto_now_add=True)
    modelo = models.CharField(max_length=120)
    precio = models.DecimalField(null=False, decimal_places=2,
max_digits=10)
```

Seguidamente, observaremos como podemos realizar modificaciones a los modelos. Cuando un modelo cambia, la tabla de la base de datos debe actualizarse de acuerdo con lo que se realizó en el

modelo. Sin embargo, si las definiciones del modelo son diferentes del esquema de la base de datos actual, se generará un error: `django.db.utils.OperationalError`.

En el modelo de historial de precio de vehículos, cambiaremos el `CharField` del modelo a un `TextField`, y agregaremos un nuevo campo llamado color con el valor predeterminado "Desconocido".

Cuando se llama al comando `"makemigrations"`, los archivos de migración subsiguientes se crearán con la hora actual del sistema adjunta al nombre del archivo. Usando el operador `--dry-run`, podemos identificar qué cambios se realizarán en el nuevo archivo de migración antes de crearlo.

```
(django-orm-postgresql)$ python manage.py makemigrations --dry-run
```

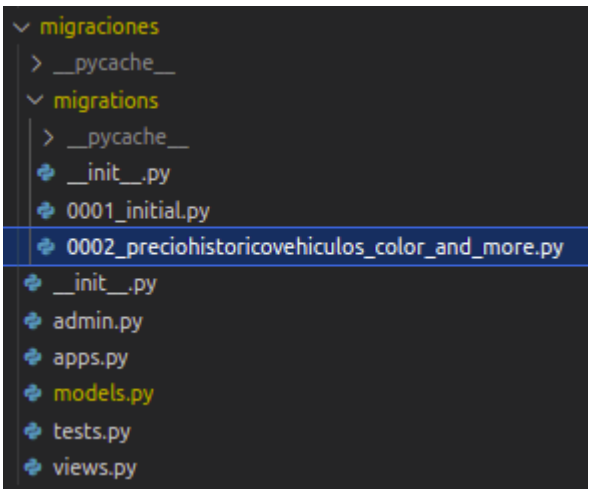
Salida:

```
Migrations for 'migraciones':
  migraciones/migrations/0002_preciohistoricovehiculos_color_and_more.py
    - Add field color to preciohistoricovehiculos
    - Alter field modelo on preciohistoricovehiculos
```

Procedemos a generar el archivo de migraciones:

```
(django-orm-postgresql)$ python manage.py makemigrations
```

El comando `makemigrations` creará un nuevo archivo llamado `0002_preciohistoricovehiculos_color_and_more.py` dentro del directorio de migraciones.



Usando el comando de migración, aplicamos los cambios a la base de datos, y los validamos explorando la base de datos desde la Shell de postgres. El comando `sqlmigrate` imprimirá los comandos SQL que se ejecutarán. Se debe tener en cuenta que el comando se puede ejecutar sin utilizar el nombre completo del archivo de migración. En este caso, se utilizan los primeros cuatro dígitos del nombre del archivo.

```
(django-orm-postgresql)$ python manage.py sqlmigrate migraciones 0002
```

Salida:

```
BEGIN;
--
-- Add field color to preciohistoricovehiculos
--
ALTER TABLE "migraciones_preciohistoricovehiculos" ADD COLUMN "color"
varchar(50) DEFAULT 'Desconocido' NOT NULL;
ALTER TABLE "migraciones_preciohistoricovehiculos" ALTER COLUMN "color"
DROP DEFAULT;
--
-- Alter field modelo on preciohistoricovehiculos
--
ALTER TABLE "migraciones_preciohistoricovehiculos" ALTER COLUMN
"modelo" TYPE text USING "modelo"::text;
COMMIT;
```

Procedemos a generar las migraciones:

```
(django-orm-postgresql)$ python manage.py migrate
```

Salida:

```
Operations to perform:
  Apply all migrations: admin, auth, blogsite, contenttypes,
migraciones, relaciones, sessions
Running migrations:
  Applying migraciones.0002_preciohistoricovehiculos_color_and_more...
OK
```

Verificamos la estructura en la base de datos:

SHELL PSQL:

```
project_orm_django=# \d migraciones_preciohistoricovehiculos
```

Salida:

```
                    Table "public.migraciones_preciohistoricovehiculos"
 Column |           Type           | Collation | Nullable |             Default
--------+--------------------------+-----------+----------+---------------------------------
 id     | bigint                   |           | not null | generated by default as identity
 fecha  | timestamp with time zone |           | not null |
 modelo | text                     |           | not null |
 precio | numeric(10,2)            |           | not null |
 color  | character varying(50)    |           | not null |
Indexes:
    "migraciones_preciohistoricovehiculos_pkey" PRIMARY KEY, btree (id)
```

El resultado anterior demuestra que los cambios realizados en el archivo del modelo se reflejan correctamente en la base de datos. El campo "modelo" se ha convertido en un campo de texto, mientras que el nuevo campo "color" se ha agregado a la base de datos.

1. LISTADO DE LAS MIGRACIONES Y REVERSIÓN DE MIGRACIONES

El comando **`showmigrations`** está diseñado para proporcionar una descripción detallada de las migraciones dentro de todas las aplicaciones asociadas con el proyecto Django dado. Si la migración ya está aplicada, se indicará con una **`"X"`** delante de la migración.

```
$ python manage. py showmigrations
```

Salida:

```
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
blogsite
 [X] 0001_initial
```

```
contenttypes
 [X] 0001_initial
 [X] 0002_remove_content_type_name
migraciones
 [X] 0001_initial
 [X] 0002_preciohistoricovehiculos_color_and_more
relaciones
 [X] 0001_initial
 [X] 0002_tipocombustible_modelocarro_tipo_combustible
 [X] 0003_directorejecutivo_delete_modelocarro_and_more
sessions
 [X] 0001_initial
```

2. REVERSIÓN DE MIGRACIONES

Django Migrations proporciona una forma conveniente de revertir las migraciones a versiones anteriores. Se puede realizar llamando al comando de **`migrate`**, más el nombre de la aplicación y el nombre de la migración antes de la migración a la que desea revertir.

```
django-admin migrate <<nombre_de_aplicación>> <<nombre_de_la_migración>>
```

El siguiente ejemplo demuestra cómo podemos retroceder desde la migración de **`0002_preciohistoricovehiculos_color_and_more.py`** (que creamos anteriormente) en la aplicación "migraciones ", a la **`migración 0001_initial.py`**.

Verificamos la estructura de la base de datos antes de realizar la reversión:

Shell de psql:

```
project_orm_django=# \d migraciones_preciohistoricovehiculos
```

Salida:

```
                    Table "public.migraciones_preciohistoricovehiculos"
 Column |           Type           | Collation | Nullable |              Default
--------+--------------------------+-----------+----------+-----------------------------------
 id     | bigint                   |           | not null | generated by default as identity
 fecha  | timestamp with time zone |           | not null |
 modelo | text                     |           | not null |
 precio | numeric(10,2)            |           | not null |
 color  | character varying(50)    |           | not null |
Indexes:
    "migraciones_preciohistoricovehiculos_pkey" PRIMARY KEY, btree (id)
```

Visualizamos el SQL que se ejecutará en la reversión a la versión `0001_inicial.py`:

```
$ python manage.py sqlmigrate migraciones 0001
```

Salida:

```
BEGIN;
--
-- Create model PrecioHistoricoVehiculos
--
CREATE TABLE "migraciones_preciohistoricovehiculos" ("id" bigint NOT
NULL PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY, "fecha" timestamp
with time zone NOT NULL, "modelo" varchar(120) NOT NULL, "precio"
numeric(10, 2) NOT NULL);
COMMIT;
```

Ejecutamos la reversión:

```
$ python manage.py migrate migraciones 0001
```

Salida:

```
Operations to perform:
  Target specific migration: 0001_initial, from migraciones
Running migrations:
  Rendering model states... DONE
  Unapplying migraciones.0002_preciohistoricovehiculos_color_and_more...
OK
```

Verificamos la estructura de la tabla Shell psql:

```
project_orm_django=# \d migraciones_preciohistoricovehiculos
```

Salida:

```
                    Table
"public.migraciones_preciohistoricovehiculos"
 Column |           Type           | Collation | Nullable |
Default
--------+--------------------------+-----------+----------+------------
----------------------
 id     | bigint                   |           | not null | generated
by default as identity
 fecha  | timestamp with time zone |           | not null |
 modelo | character varying(120)   |           | not null |
 precio | numeric(10,2)            |           | not null |
Indexes:
    "migraciones_preciohistoricovehiculos_pkey" PRIMARY KEY, btree
(id)" varchar(120) NOT NULL, "precio" numeric(10, 2) NOT NULL);
COMMIT;
```

Verificamos las migraciones:

```
$ python manage.py showmigrations
```

Salida:

```
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
blogsite
 [X] 0001_initial
contenttypes
 [X] 0001_initial
 [X] 0002_remove_content_type_name
migraciones
 [X] 0001_initial
```

```
 [ ] 0002_preciohistoricovehiculos_color_and_more
relaciones
 [X] 0001_initial
 [X] 0002_tipocombustible_modelocarro_tipo_combustible
 [X] 0003_directorejecutivo_delete_modelocarro_and_more
sessions
 [X] 0001_initial
```

Hay algo importante que debe recordarse al revertir las migraciones. Aunque la mayoría de las operaciones de la base de datos se pueden revertir, si se elimina un campo del modelo y las éstas se aplican a la base de datos, se eliminarán todos los datos dentro de la columna. Revertir en esta instancia no recuperará los datos, sólo creará el campo eliminado.

NOMBRES A LAS MIGRACIONES:

En el ejemplo anterior permitimos que Django administre la conversión de nombres de los archivos de migración. Sin embargo, éste brinda la opción de crear nombres personalizados usando el operador **"--name"**.

En el siguiente ejemplo eliminaremos el archivo de la migración **0002_preciohistoricovehiculos_color_and_more.py**, y lo volveremos a crear con el nombre personalizado **"cambiando_modelo_a_text_y_agregando_campo_color"**. Procedemos a eliminar el archivo **0002_preciohistoricovehiculos_color_and_more.py**.

Creamos el nuevo archivo de migración:

```
$ python manage.py makemigrations migraciones --name
cambiando_modelo_a_text_y_agregando_campo_color
```

Salida:

```

Migrations for 'migraciones':

migraciones/migrations/0002_cambiando_modelo_a_text_y_agregando_campo_co
lor.py
    - Add field color to preciohistoricovehiculos
    - Alter field modelo on preciohistoricovehiculos python manage.py
makemigrations migraciones --name
cambiando_modelo_a_text_y_agregando_campo_color
```

Comprobamos las migraciones:

```
$ python manage.py showmigrations
```

Salida:

```
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
blogsite
 [X] 0001_initial
contenttypes
 [X] 0001_initial
 [X] 0002_remove_content_type_name
migraciones
 [X] 0001_initial
 [ ] 0002_cambiando_modelo_a_text_y_agregando_campo_color
relaciones
 [X] 0001_initial
 [X] 0002_tipocombustible_modelocarro_tipo_combustible
 [X] 0003_directorejecutivo_delete_modelocarro_and_more
sessions
 [X] 0001_initial
```

Finalmente, ya aprendimos cómo usar de manera efectiva las migraciones en un proyecto de Django. La migración es una forma de crear y modificar tablas en una base de datos que ofrece el Framework Django. El uso de migraciones con modelos permite a los desarrolladores renunciar al uso de SQL para manipular una base de datos, al menos que se trate de casos específicos de uso avanzado. Esto reduce la carga de trabajo de los desarrolladores, y facilita la depuración de aplicaciones. Además, permitir la reversión de las migraciones actúa como un sistema de control de versiones para la base de datos que ayuda a los desarrolladores a revertir los cambios en caso de error o con fines de prueba.