Django y su integración con bases de datos

Django y su integración con base de datos

{desafío} latam_

¿Qué aprenderemos en este módulo?

Construir aplicaciones web que manipulan datos en una base de datos SQL utilizando Python/Django y las componentes que el lenguaje dispone para su uso para dar solución a un requerimiento.

{desafío} latam_

Describir las características fundamentales de la integración del framework Django con bases de datos.

Reconocer las aplicaciones preinstaladas con el motor Django distinguiendo su utilidad como apoyo al desarrollo.

- Unidad 1:
  Django y su integración con bases de datos
- Unidad 2:
  Modelos de datos y el ORM de Django
- Unidad 3:
  Proyecto

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Reconoce las características del framework Django aplicado a las bases de datos.*
- *Reconoce los paquetes de instalación de base de datos en Django.*
- *Reconoce las características de Django como ORM para su integración con una base de datos.*

{desafío} latam_

¿Cuáles eran algunas de las características de Django?

/* Conectando a una base de datos */

Conectando a una base de datos

Para comenzar, luego de crear un proyecto, o trabajando ya sobre un proyecto existente, vamos a revisar en el archivo de configuraciones (***settings.py***) la sección de la base de datos.

Aquí encontramos la variable de configuración "**DATABASES**", que es un diccionario que contiene la clave "default" y cuyo valor es otro diccionario que trae la configuración de la base de datos correspondiente a la clave

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

Conectando a una base de datos

Cuando vamos a configurar motores de bases de datos más complejos como postgres o mysql, necesitamos realizar configuraciones adicionales.
Acá, podemos ver un bloque de código que nos muestra la configuración para un motor de bases de datos postgres, que contiene los parámetros para este tipo de motores de bases de datos.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'mydatabaseuser',
        'PASSWORD': 'mypassword',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

Conectando a una base de datos

Parámetros de configuración

De los parámetros utilizados para la configuración podemos definir:

- **default**: Es la base de datos utilizada por defecto por el sistema. Al configurar más de una base de datos podemos dejar la configuración por defecto en blanco, dependiendo de nuestras necesidades.

- **ENGINE**: En esta configuración debemos seleccionar el motor correspondiente a nuestra base de datos.
  - 'django.db.backends.postgresql'
  - 'django.db.backends.mysql'
  - 'django.db.backends.sqlite3'
  - 'django.db.backends.oracle'

Conectando a una base de datos

Parámetros de configuración

- **NAME**: El nombre de la base de datos que utilizaremos.
- **USER**: Nuestro usuario para acceder a la base de datos.
- **PASSWORD**: La contraseña de nuestro usuario de base de datos.
- **HOST**: La URL o la dirección IP del servidor donde tenemos nuestra base de datos, en el ejemplo está en nuestro equipo local, por lo tanto es la IP 127.0.0.1 que es la ip por defecto de nuestro equipo.
- **PORT**: El puerto por el cual el servidor de bases de datos pública el servicio al exterior, en este caso, por defecto Posgresql utiliza el puerto 5432 y Mysql utiliza el 3306.

Conectando a una base de datos

Instalar controladores

A parte de realizar este configuración en el archivo **settings.py**, debemos instalar el controlador o driver que es el programa que nos permite “comunicarnos” con la base de datos y su dialecto particular. Los casos más comunes son los conectores para mysql y postgresql. La instalación la realizamos de la siguiente forma:

```
# para mysql utilizamos el conector mysqlclient
pip install mysqlclient
# para postgresql utilizamos el conector psycopg2
pip install psycopg2
```

/* Bases de datos sql soportadas por Django: MySQL, Postgresql, SQLite, Oracle */

Las bases de datos soportadas por Django

Bases de datos soportadas por Django

MySQL

Este motor de bases de datos relacionales es uno de los más populares en la actualidad, especialmente en el desarrollo web.
Dentro de sus ventajas se encuentran las siguientes:

- Es multiplataforma y con una gran cantidad de instaladores y conectores para varios sistemas operativos y lenguajes de programación.
- Es bastante segura para ser desplegada en ambientes productivos.
- Se basa en el estándar SQL

Bases de datos soportadas por Django

Conexión a MySQL

Para la conexión a una base de datos de MySQL, se deben seguir los siguientes pasos:

- Para esto abriremos un terminal y correremos nuestro comando como se muestra en la Figura 1.
- Por último, hay que modificar la variable ***DATABASES*** como se indica en la Figura 2 en el archivo settings.py de nuestra aplicación. Ademas, se debe cambiar los valores marcados en amarillo de la figura correspondiente a las credenciales de tu servidor de Base de datos.

```
pip install mysqlclient
```

Figura 1: Comando para instalación de cliente MySql

```
DATABASES = {
    'default': {
        'ENGINE':
'django.db.backends.mysql',
        'NAME': 'mydatabase',
        'USER': 'mydatabaseuser',
        'PASSWORD': 'mypassword',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

Figura 2: Variable de conexión a MySQL

Bases de datos soportadas por Django

PostgreSQL

Es un poderoso motor de bases de datos relacionales con más de 30 años de desarrollo. Provee una herramienta de administración y desarrollo basada en web llamada pgAdmin.

Según la encuesta 2020 del sitio StackOverflow, Postgresql es el segundo sistema de bases de datos más popular, solo debajo de MySQL. La razón de su popularidad podría deberse a:

- Es libre y open source, siendo completamente gratuito su uso tanto a nivel personal como empresarial.
- Posee una larga lista de tipos de datos soportados, como JSON, XML (formatos de archivos para intercambio de datos), entre otros.
- Es muy rápida y segura.
- Es fácil migrar a ella desde otros motores.
- Adhiere en gran parte al estándar SQL.

PostgreSQL

Bases de datos soportadas por Django

Conexión a Postgre

Similar a MySQL, para la conexión a una base de datos de Postgres se tiene que seguir los siguientes pasos:

- Para esto abriremos un terminal y correremos nuestro comando como se muestra en la Figura 1.
- Posteriormente, hay que modificar la variable DATABASES como se indica en la Figura 2 en el archivo settings.py de nuestra aplicación. Ademas de cambiar los valores marcados en amarillo de la figura correspondiente a las credenciales de tu servidor de Base de datos.

```
pip install psycopg2
```

Figura 1: Comando para instalación de cliente Postgres

```
DATABASES = {
    'default': {
        'ENGINE':
'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'mydatabaseuser',
        'PASSWORD': 'mypassword',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

Figura 2: Variable de conexión a Postgre

Bases de datos soportadas por Django

SQLite

Es una librería que implementa un motor de bases de datos SQL (relacional) pequeño, rápido, auto-contenido, de alta disponibilidad y con todas las funciones.

Django trae soporte por defecto para este motor de bases de datos, y no es necesario realizar alguna instalación anexa, tanto de este como de un conector.

Los datos son guardados por defecto en la raíz del proyecto en un archivo llamado db.sqlite3.

Bases de datos soportadas por Django

Oracle

Este motor de bases funciona con un licenciamiento Propietario y tiene varios modelos de negocio como instalación local y en la nube, y en comparación con Postgres tiene ventaja en aspectos como seguridad, disponibilidad, replicación y transacciones por segundo, pero, debido a su modelo de negocio y su licenciamiento, muchas de estas capacidades agregan costo extra a la suscripción final.

Bases de datos soportadas por Django

Bases de datos NoSql

El nombre NoSQL originalmente se atribuía a bases de datos No Sql (no relacionales) aunque últimamente en algunos sitios lo llaman Not Only SQL (No solo Sql), lo cual puede englobar una serie de sistemas de administración de bases de datos que almacenan la información de forma flexible de acuerdo al requerimiento.

Ejemplo de estas son:

- MongoDB
- Redis
- CassandraDB

Bases de datos soportadas por Django

Conexión a NoSQL

Oficialmente Django no soporta este tipo de bases de datos, pero estas han generado los conectores necesarios para poder ser utilizadas.
En el caso de MongoDB se debe realizar los siguientes pasos:

- Instalar Djongo vía pip (Figura 1).
- Por último hay que modificar la variable DATABASES como se indica en la Figura 2 en el archivo settings.py de nuestra aplicación.
- Por último realizar una migración.

```
pip install djongo
```

Figura 1: Comando para instalación de cliente Djongo

```
DATABASES = {
        'default': {
                'ENGINE': 'djongo',
                'NAME': 'your-db-name',
        }
}
```

Figura 2: Variable de conexión a MongoDB

```
python manage.py makemigrations
python manage.py migrate
```

Figura 3: Comando para realizar las migraciones

/* Introducción al ORM de Django */

{desafío}
latam_

Introducción al ORM de Django

Una de las más poderosas características de Django es su Object-Relational Mapper, más conocido como ORM (Mapeador relacional de objetos.).

Un ORM no es una característica exclusiva de Django, Es más una metodología de trabajo que se implementa mediante distintos software para los lenguajes de programación más demandados y sus frameworks.

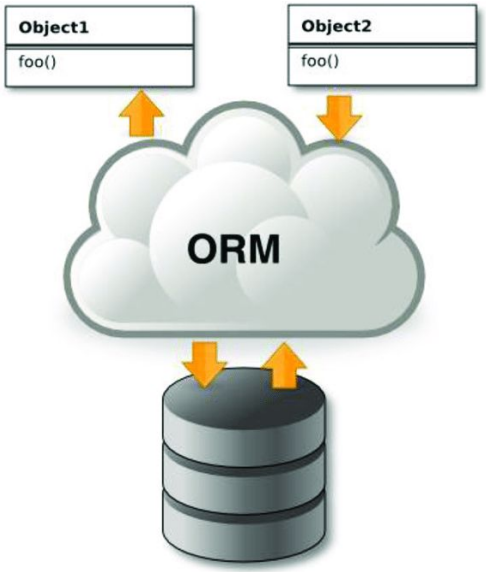
Un ORM mapea el conjunto de tablas y relaciones de una base de datos relacional y genera una capa de abstracción de cara al desarrollador, donde nos evita trabajar directamente con sentencias de SQL y en cambio, nos presenta una serie de objetos con una cantidad definida de métodos, que nos permiten realizar casi cualquier operación SQL, pero sin dejar nuestro lenguaje de programación de elección.

Introducción al ORM de Django

Una de las más poderosas características de Django es su Object-Relational Mapper, más conocido como ORM (Mapeador relacional de objetos.).

Un ORM no es una característica exclusiva de Django, Es más una metodología de trabajo que se implementa mediante distintos software para los lenguajes de programación más demandados y sus frameworks.

Introducción al ORM de Django



Fuente: *Researchgate*

Un ORM mapea el conjunto de tablas y relaciones de una base de datos relacional y genera una capa de abstracción de cara al desarrollador, donde nos evita trabajar directamente con sentencias de SQL y en cambio, nos presenta una serie de objetos con una cantidad definida de métodos, que nos permiten realizar casi cualquier operación SQL, pero sin dejar nuestro lenguaje de programación de elección.

El usar ORM también permite separar la lógica de la herramienta, por lo tanto si en algún momento se necesita cambiar el gestor de BD, basta con cambiar el motor en el archivo settings.py y el resto del código no se ve afectado.

Introducción al ORM de Django

Por ejemplo, realizamos una búsqueda en la tabla de Alumnos, donde queremos obtener los alumnos de apellido Perez.

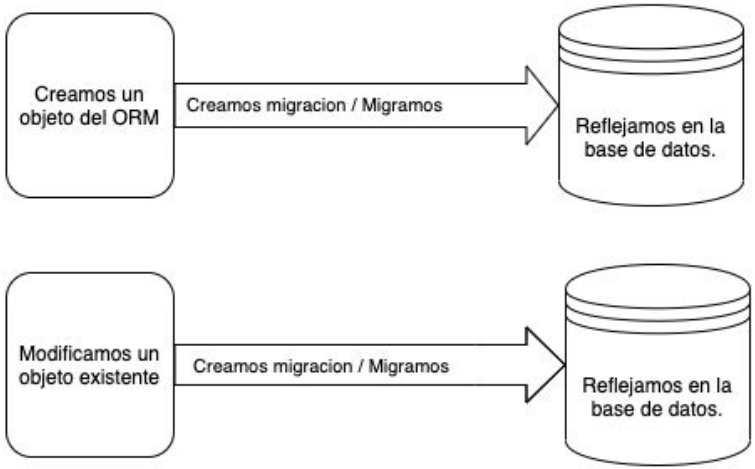
```
En SQL Tradicional:

SELECT * FROM Alumnos WHERE apellido = 'PEREZ'

Utilizando el ORM de django.

resultados = Alumnos.object.filter(apellido='PEREZ').all()
```

/* Introducción a las migraciones */

Introducción a las migraciones

Las migraciones de Django son una funcionalidad que nos permite manejar un control de versiones de los elementos de nuestra base de datos.

Por ejemplo, nosotros tenemos una serie de modelos de datos (models.py) creados en nuestro proyecto Django, al realizar la migración por primera vez, este proceso refleja los modelos del proyecto en elementos de la base de datos, como tablas, relaciones, etc.



Proceso de Migración Django
***Fuente**: Desafío Latam.*

{desafío} latam_

/* Ejecutando SQL Queries en Django */

Ejecutando SQL Queries en Django

Django nos provee un método para la ejecución directa de sentencias de SQL mediante un objeto de conexión, el cual apunta a la base de datos que configuramos previamente en settings.py.

Para hacer esto:

- Importamos la librería django.db.connection
- Obtenemos un objeto cursor que nos entrega la conexión.
- Luego con este objeto podemos ejecutar las consultas, mediante el método execute.

Ejecutando SQL Queries en Django

Para recuperar información en caso que hayamos realizado un select, podemos utilizar los métodos **cursor.fetchone()** o **cursor.fetchall()**.
Para pasar parámetros a una query, lo realizamos utilizando los caracteres %s en el string de la consulta y luego pasamos una lista de parámetros en el método execute.

```
from django.db import connection

def my_custom_sql(self):
    with connection.cursor() as cursor:
        cursor.execute("UPDATE bar SET foo = 1 WHERE baz = %s", [self.baz])
        cursor.execute("SELECT foo FROM bar WHERE baz = %s", [self.baz])
        row = cursor.fetchone()

  return row
```

Ejecutando SQL Queries en Django

También podemos llamar procedimientos almacenados de una forma similar a como lo hicimos con las queries con parametros, pero utilizando el metodo **cursor.callproc()**

```
with connection.cursor() as cursor:
    cursor.callproc('test_procedure', [1, 'test'])
```

Ejercicio guiado:

Instalación y ejecución de migraciones

Instalación y ejecución de migraciones

Ejercicio guiado

En el presente ejercicio realizaremos la instalación y ejecutaremos las migraciones de un proyecto nuevo utilizando una base de datos sqlite.

1. Crear un entorno virtual (puede usar venv u otro método que conozca) y luego creamos un proyecto nuevo llamado **proyecto_capitulo_1**, en este caso utilizamos la versión 3.2.5 de django.

```
python -m venv .\venv
.\venv\Scripts\activate.bat

pip install django
django-admin startproject
proyecto_capitulo_1
cd proyecto_capitulo_1
```

Instalación y ejecución de migraciones

Ejercicio guiado

2. Creamos una aplicación llamada testadl

```
python manage.py startapp testadl
```

Instalación y ejecución de migraciones

Ejercicio guiado

3. Agregamos testadl a las aplicaciones instaladas en django.

   Para esto abrimos el proyecto con VSCODE u otro editor de texto de su elección, nos dirigimos a **settings.py** y agregamos la línea correspondiente.

```
31  # Application definition
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      'testadl'
41  ]
```

Instalación y ejecución de migraciones

Ejercicio guiado

4. Vamos a crear un modelo básico.

   Con el editor de texto abrimos el archivo **testadl\models.py**

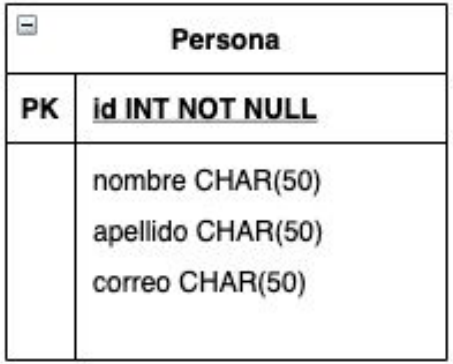
```
from django.db import models

class Persona(models.Model):
    id = models.AutoField(primary_key=True)
    nombre = models.CharField(max_length=50)
    apellido = models.CharField(max_length=50)
    correo = models.EmailField(max_length=50)
```

Instalación y ejecución de migraciones

Ejercicio guiado

5. Aplicamos las migraciones desde la terminal, por el momento correremos estos dos comandos, los cuales serán explicados con mayor detalle más adelante

```
python manage.py makemigrations
python manage.py migrate
```



Modelo una vez creado en la base de datos
***Fuente**: Desafío Latam.*

Instalación y ejecución de migraciones

Ejercicio guiado

6. Entramos a la shell de django desde la terminal

```
python manage.py shell
```

Instalación y ejecución de migraciones

Ejercicio guiado

7. Ejecutaremos las siguientes sentencias para agregar dos Personas (se insertarán 2 registros en la tabla Persona de la base de datos.).

```
>>> from testadl.models import Persona
>>> p1 = Persona(nombre='John', apellido='Doe', correo='jdoe@mail.com')
>>> p1.save()
>>> p2 = Persona(nombre='Chuck', apellido='Norris', correo='chuck@mail.com')
>>> p2.save()
```

¿Qué es el ORM de Django?

¿Cuáles son los motores de base de datos soportados por Django?

¿Django permite la ejecución de Queries de SQL directamente?

¿Qué es una migración?

{desafío} latam_

Próxima sesión...

- *Desafío guiado*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales