**¿Cuáles son las bases de datos soportadas por Django y en qué se diferencian?**

1. SQLite: Es el motor de bases de datos predeterminado en Django. Se almacena como un archivo en el sistema de archivos y no requiere una configuración de servidor separada. Es ligero y adecuado para aplicaciones de desarrollo y pruebas o para aplicaciones con bajo tráfico.

2. PostgreSQL: Es un potente sistema de gestión de bases de datos relacional y de código abierto. Proporciona características avanzadas, como soporte para transacciones, integridad referencial y consultas complejas. PostgreSQL es conocido por su rendimiento, escalabilidad y capacidad de manejar grandes volúmenes de datos.

3. MySQL / MariaDB: Son sistemas de gestión de bases de datos relacionales ampliamente utilizados y de código abierto. Tienen características similares y son conocidos por su estabilidad, rendimiento y escalabilidad.

4. Oracle: Es un sistema de gestión de bases de datos relacional de propósito general. Es conocido por su escalabilidad, seguridad y soporte para aplicaciones empresariales de alto rendimiento. Django ofrece soporte para Oracle, lo que permite a los desarrolladores utilizar Django en entornos empresariales y con bases de datos Oracle existentes.

**¿Qué es una migración en Django y para qué se utiliza?**

En Django, una migración es un archivo generado automáticamente que describe los cambios en la estructura de la base de datos. Las migraciones se utilizan para realizar cambios en el esquema de la base de datos de manera controlada y mantenerlo sincronizado con los modelos de Django.

Las migraciones tienen varios propósitos y beneficios:

1. Creación y actualización de tablas

2. Control de versiones del esquema

3. Portabilidad y flexibilidad.

4. Preservación de datos

**¿Cuál es la diferencia entre usar consultas SQL y consultas ORM en Django?**

Consultas SQL: En Django, puedes utilizar consultas SQL directamente para interactuar con la base de datos. Esto implica escribir sentencias SQL en tu código para realizar operaciones como selección, inserción, actualización y eliminación de datos.

* Control total sobre las sentencias SQL y su ejecución.
* Útil para consultas complejas y personalizadas que pueden ser difíciles de expresar con consultas ORM.
* Puede aprovechar características específicas de la base de datos subyacente.

Consultas ORM: Django proporciona una capa de abstracción llamada ORM que permite interactuar con la base de datos utilizando un enfoque orientado a objetos. En lugar de escribir consultas SQL directamente, utilizas los modelos de Django y sus métodos para realizar operaciones en la base de datos.

Ventajas de las consultas ORM:

* Sintaxis más legible y expresiva que refleja la estructura del modelo de datos.
* Generación automática de consultas SQL según las operaciones realizadas.
* Funciones y métodos incorporados para realizar operaciones comunes, como filtrar, ordenar y unir datos.
* Mitiga riesgos de seguridad al escapar automáticamente los parámetros de la consulta.

**¿Cómo se instalan los paquetes de base de datos en Django y cuál es su importancia?**

Instalar el paquete de la base de datos: Dependiendo del motor de base de datos que se desee utilizar, se deberá instalar el paquete correspondiente. Por ejemplo, si utilizamos PostgreSQL como base de datos, debemos instalar el paquete psycopg2.

Configurar la conexión a la base de datos en Django: Una vez instalado el paquete de la base de datos, debes configurar la conexión en el archivo settings.py de tu proyecto Django. Allí especificarán los detalles de conexión, como el motor de base de datos, el nombre de la base de datos, el usuario y la contraseña. Por ejemplo, para configurar PostgreSQL, puedes tener el siguiente código en tu archivo settings.py:

DATABASES = { 'default':

{ 'ENGINE': 'django.db.backends.postgresql',

'NAME': 'mydatabase',

'USER': 'myuser',

'PASSWORD': 'mypassword',

'HOST': 'localhost',

'PORT': '5432', }

}

Realizar las migraciones: Después de configurar la conexión a la base de datos, debes ejecutar las migraciones para crear las tablas correspondientes en la base de datos. Utilizando el comando python manage.py migrate, Django creará las tablas necesarias en la base de datos.

La importancia de los paquetes de base de datos en Django radica en que son los componentes fundamentales para interactuar con una base de datos subyacente. Estos paquetes permiten a Django establecer una conexión con la base de datos, ejecutar consultas, realizar operaciones CRUD (crear, leer, actualizar, eliminar) en los datos y mantener la sincronización entre los modelos de Django y la estructura de la base de datos.

**¿Qué ventajas ofrece Django como ORM para la integración con una base de datos?**

1. Abstracción de base de datos: Django proporciona una capa de abstracción que permite interactuar con la base de datos utilizando modelos y consultas orientadas a objetos en lugar de escribir consultas SQL directamente. Esta abstracción facilita el desarrollo al eliminar la necesidad de conocer detalles específicos de la sintaxis de SQL para cada motor de base de datos compatible.

2. Sintaxis legible y expresiva: Las consultas ORM en Django utilizan una sintaxis legible y expresiva que refleja la estructura del modelo de datos. Esto hace que el código sea más fácil de entender y mantener, ya que las consultas se escriben en términos de objetos y campos en lugar de manipular directamente las tablas y las columnas de la base de datos.

3. Generación automática de consultas SQL: Django genera automáticamente las consultas SQL correspondientes según las operaciones realizadas a través del ORM. Esto elimina la necesidad de escribir consultas SQL manualmente, lo que ahorra tiempo y reduce la posibilidad de cometer errores sintácticos en las consultas.

4. Funciones y métodos integrados: Django ORM proporciona una amplia gama de funciones y métodos integrados para realizar operaciones comunes en la base de datos, como filtrar, ordenar, unir tablas y realizar agregaciones. Estas funciones y métodos facilitan la escritura de consultas complejas y reducen la cantidad de código necesario.

5. Seguridad incorporada: El ORM de Django maneja automáticamente la escapada de parámetros de consulta para prevenir inyecciones de SQL y otros ataques relacionados con la seguridad. Esto ayuda a mitigar riesgos de seguridad y evita la necesidad de escapar manualmente los valores de los parámetros en las consultas.

6. Portabilidad de la base de datos: Django ORM permite desarrollar aplicaciones que pueden cambiar fácilmente entre diferentes motores de base de datos compatibles, como PostgreSQL, MySQL, SQLite, entre otros. Esto brinda flexibilidad y portabilidad a tu aplicación, ya que no estás atado a un motor de base de datos específico.

DATABASES = {,

}

REPOSITORIO:

https://github.com/mtapiaoses/Pindividual\_modulo7