MÓDULO 7 – ABP1 – INDIVIDUAL 1

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

RESPUESTA:

* SQLite: Es una base de datos liviana y sin servidor que se almacena en un archivo local. Es útil para aplicaciones pequeñas o en desarrollo, ya que no requiere configuración de servidor adicional. Sin embargo, puede tener limitaciones en cuanto a escalabilidad y rendimiento en aplicaciones más grandes.
* PostgreSQL: Es una base de datos relacional de código abierto conocida por su robustez, escalabilidad y capacidad de manejar grandes volúmenes de datos. Django proporciona un buen soporte para características avanzadas de PostgreSQL, como tipos de datos personalizados, consultas complejas y funciones específicas.
* MySQL/MariaDB: Estas bases de datos relacionales son ampliamente utilizadas en entornos web. Django es compatible con ambas y proporciona características como replicación, clústeres y gestión de transacciones. Sin embargo, la compatibilidad específica puede variar según la versión de Django y la biblioteca de Python utilizada.
* Oracle: Django también admite la base de datos Oracle, una opción común en entornos empresariales. Se requiere una configuración adicional para conectarse a Oracle, como la instalación del controlador de base de datos apropiado y la configuración de las variables de entorno.
* Microsoft SQL Server: Django admite Microsoft SQL Server, lo que permite el desarrollo de aplicaciones web utilizando esta base de datos relacional. Al igual que Oracle, se necesitan pasos adicionales de configuración para establecer la conexión.

Independientemente de la base de datos que se utilice, Django ofrece una capa de abstracción que permite escribir código de manera independiente de la base de datos subyacente. Esto facilita el cambio entre diferentes bases de datos sin tener que realizar modificaciones significativas en el código de la aplicación.

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

RESPUESTA:

Una migración es un proceso que permite realizar cambios en el esquema de la base de datos de manera controlada y consistente. Se utiliza para mantener sincronizado el modelo de datos de la aplicación con la estructura de la base de datos subyacente.

Una migración en Django se basa en el concepto de "modelo" que define la estructura de los datos en la aplicación. Cuando se realizan cambios en el modelo, como agregar, eliminar o modificar campos de una tabla, Django utiliza las migraciones para aplicar esos cambios en la base de datos.

* Creación inicial de la base de datos: Cuando se inicia un nuevo proyecto de Django, se pueden definir los modelos que representan las tablas de la base de datos. Las migraciones se utilizan para crear esas tablas iniciales en la base de datos.
* Cambios en el esquema de la base de datos: A medida que se desarrolla una aplicación, es posible que se necesite modificar la estructura de la base de datos. Las migraciones permiten realizar cambios como agregar nuevos campos a una tabla existente, cambiar el tipo de datos de un campo, eliminar campos o incluso crear nuevas tablas.
* Versionado del esquema: Las migraciones en Django también ayudan a mantener un historial de los cambios realizados en el esquema de la base de datos a lo largo del tiempo. Cada migración se guarda en un archivo que registra los cambios específicos realizados en el modelo. Esto facilita el seguimiento de los cambios, la reversión de migraciones anteriores y la actualización de la base de datos de manera incremental.
* Despliegue en diferentes entornos: Las migraciones permiten que el esquema de la base de datos se mantenga sincronizado en diferentes entornos, como el desarrollo, pruebas y producción. Esto garantiza que todos los cambios realizados en el modelo se apliquen de manera consistente en cada entorno.

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

La diferencia principal entre usar consultas SQL y consultas ORM (Object-Relational Mapping) en Django radica en la forma en que se interactúa con la base de datos.

Consultas SQL:

* + Las consultas SQL implican escribir sentencias SQL directamente para interactuar con la base de datos.
  + Requieren conocimiento de SQL y su sintaxis.
  + Proporcionan un control más granular sobre las consultas y permiten realizar operaciones avanzadas y personalizadas en la base de datos.
  + Las consultas SQL pueden ser más rápidas en ciertos casos cuando se requieren operaciones complejas y optimizadas.
  + Sin embargo, el código puede volverse más largo y propenso a errores si se escriben muchas consultas SQL personalizadas.

Consultas ORM:

* Las consultas ORM en Django se basan en el concepto de abstracción de base de datos y permiten interactuar con la base de datos utilizando objetos y métodos proporcionados por el ORM de Django.
* Se utilizan métodos y funciones de Django para realizar operaciones CRUD (crear, leer, actualizar, eliminar) en los modelos y las tablas de la base de datos.
* No es necesario escribir sentencias SQL directamente, ya que el ORM se encarga de generar las consultas SQL correspondientes según las operaciones realizadas.
* Las consultas ORM son más fáciles de escribir y entender, especialmente para desarrolladores que no están familiarizados con SQL.
* Proporcionan portabilidad entre diferentes bases de datos compatibles con Django, ya que el ORM se encarga de generar el código SQL específico para cada base de datos.
* Las consultas ORM pueden ser menos eficientes en términos de rendimiento en comparación con consultas SQL personalizadas en casos de operaciones complejas o consultas masivas de datos.

En general, la elección entre usar consultas SQL o consultas ORM en Django depende del escenario y los requisitos del proyecto. Si se necesitan operaciones personalizadas o complejas en la base de datos, las consultas SQL pueden ser más adecuadas. Por otro lado, si se busca una forma más sencilla y portátil de interactuar con la base de datos, las consultas ORM proporcionadas por Django son una opción conveniente.

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

RESPUESTA:

En Django, los paquetes de base de datos se instalan utilizando el sistema de gestión de paquetes de Python, como pip. Los paquetes de base de datos son necesarios para permitir la conexión y la interacción con una base de datos específica desde Django.

A continuación, se muestra el proceso general para instalar los paquetes de base de datos en Django:

* Asegúrate de tener Python y pip instalados en tu sistema.
* Abre una terminal o línea de comandos y ejecuta el siguiente comando para instalar el paquete de base de datos deseado. Por ejemplo, si quieres utilizar PostgreSQL, el comando sería: pip install psycopg2
* Una vez que se ha instalado el paquete de base de datos, debes asegurarte de configurar correctamente la conexión a la base de datos en el archivo de configuración de Django (settings.py). Esto incluye proporcionar el nombre del paquete de base de datos instalado, el nombre de usuario, la contraseña, el host y otros detalles de configuración necesarios para establecer la conexión.

La importancia de los paquetes de base de datos en Django radica en que permiten a la aplicación interactuar con una base de datos específica. Estos paquetes proporcionan los controladores necesarios y las interfaces para realizar operaciones de lectura y escritura en la base de datos desde Django. Sin el paquete de base de datos adecuado instalado y configurado, Django no podrá establecer una conexión con la base de datos y, por lo tanto, no podrá almacenar ni recuperar datos de manera efectiva.

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

RESPUESTA:

Django ofrece varias ventajas como ORM (Object-Relational Mapping) para la integración con una base de datos:

* Abstracción de la base de datos: Django proporciona una capa de abstracción que permite interactuar con la base de datos sin tener que escribir consultas SQL directamente. En lugar de eso, se utilizan objetos y métodos de Django para realizar operaciones CRUD (crear, leer, actualizar, eliminar) en los modelos y tablas de la base de datos. Esta abstracción facilita el desarrollo al evitar la necesidad de conocer en profundidad la sintaxis y particularidades de cada base de datos subyacente.
* Independencia de la base de datos: El ORM de Django permite escribir código independiente de la base de datos. Esto significa que se puede desarrollar una aplicación utilizando Django y, en teoría, cambiar la base de datos subyacente sin tener que modificar gran parte del código. Django proporciona soporte para múltiples bases de datos, lo que permite utilizar diferentes sistemas de gestión de bases de datos (como PostgreSQL, MySQL, SQLite, etc.) con la misma aplicación.
* Portabilidad de la aplicación: La abstracción de base de datos de Django y su soporte para múltiples bases de datos facilitan la portabilidad de la aplicación entre diferentes entornos. Esto significa que se puede desarrollar una aplicación en un entorno de desarrollo utilizando una base de datos determinada, y luego desplegarla en un entorno de producción con una base de datos diferente sin tener que realizar cambios significativos en el código.
* Seguridad: Django ayuda a prevenir ataques de seguridad comunes en las operaciones de la base de datos. Por ejemplo, protege contra inyección de SQL, asegurando que las consultas sean generadas de manera segura. Django también ofrece mecanismos de autenticación y autorización integrados para proteger los datos de la aplicación.
* Funcionalidades avanzadas: Django ORM proporciona una amplia gama de funcionalidades avanzadas, como consultas complejas, agregaciones, filtrado, ordenamiento, consultas enlazadas y más. También permite definir relaciones entre modelos, como relaciones uno a uno, uno a muchos y muchos a muchos, y realizar operaciones en cascada en esas relaciones.
* Migraciones automáticas: Django ofrece un sistema de migraciones que facilita la evolución del esquema de la base de datos a medida que la aplicación se desarrolla. Las migraciones permiten realizar cambios en el modelo de datos y aplicarlos a la base de datos de manera controlada y consistente, manteniendo un historial de cambios y facilitando la reversión de migraciones anteriores.