Franco Fuentes Soto – Modulo 7 - Individual 1

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

SQLite: Es una base de datos ligera y de archivo único que se integra directamente con Django. Es fácil de configurar y adecuada para aplicaciones de desarrollo y pruebas, así como para aplicaciones pequeñas o de bajo tráfico.

PostgreSQL: Es una base de datos relacional de código abierto y robusta. Ofrece soporte para características avanzadas como integridad referencial, transacciones, replicación y funciones avanzadas de búsqueda. PostgreSQL es una opción popular para aplicaciones de alto rendimiento y escalabilidad.

MySQL y MariaDB: Son bases de datos relacionales ampliamente utilizadas y de código abierto. Ofrecen una buena combinación de rendimiento y funcionalidad. MySQL ha sido durante mucho tiempo la opción preferida, mientras que MariaDB es un fork de MySQL que mantiene compatibilidad con el mismo.

Oracle: Es una base de datos empresarial con capacidades avanzadas y escalabilidad. Django proporciona soporte para Oracle, lo que permite desarrollar aplicaciones utilizando Oracle como backend de base de datos.

Microsoft SQL Server: Es una base de datos relacional de Microsoft con un conjunto completo de características y herramientas para desarrolladores. Django tiene soporte para Microsoft SQL Server, lo que permite utilizarlo como base de datos subyacente en aplicaciones Django.

¿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 mantener la consistencia entre el modelo de datos definido en el código de Django y la estructura real de la base de datos subyacente.

Cuando se realizan cambios en los modelos de Django, como agregar una nueva tabla, modificar una columna existente o crear una relación entre tablas, las migraciones se encargan de aplicar esos cambios en la base de datos de manera controlada y segura, sin perder los datos existentes.

Las migraciones en Django facilitan el manejo de los cambios en la estructura de la base de datos a medida que evoluciona una aplicación. Permiten aplicar cambios de manera incremental y consistente, asegurando que la base de datos esté siempre sincronizada con los modelos definidos en el código de Django.

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

La diferencia principal entre el uso de consultas SQL y consultas ORM (Object-Relational Mapping) en Django radica en cómo se realiza la interacción con la base de datos y cómo se representa la información en el código.

1. Consultas SQL:
    - Las consultas SQL se escriben directamente en el lenguaje SQL (Structured Query Language).
    - Requieren conocimientos de SQL y la sintaxis específica de la base de datos utilizada.
    - Se deben construir manualmente las consultas SQL, lo que puede llevar a un código más extenso y propenso a errores.
    - Ofrecen un control preciso sobre las consultas y permiten aprovechar características específicas de la base de datos.

- Pueden ser más eficientes en ciertos casos, especialmente cuando se requieren consultas complejas o altamente optimizadas.

2. Consultas ORM:
    - Las consultas ORM se realizan a través de un mapeo objeto-relacional, donde las consultas se construyen utilizando métodos y atributos de los modelos de Django.
    - Utilizan una interfaz más orientada a objetos, donde las tablas de la base de datos se representan como clases y las filas de la tabla como instancias de esas clases.
    - Proporcionan una abstracción de nivel superior, lo que facilita la escritura de consultas sin necesidad de conocer SQL en detalle.
    - Permiten un código más legible y mantenible, ya que se basan en métodos y atributos propios del ORM de Django.
    - La sintaxis de las consultas ORM de Django es independiente de la base de datos subyacente, lo que brinda mayor portabilidad.

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

En Django, los paquetes de base de datos se instalan como dependencias adicionales según la base de datos que planeas utilizar. La instalación de estos paquetes es necesaria para que Django pueda interactuar con la base de datos específica.

A continuación, se muestra cómo instalar los paquetes de base de datos para algunas de las bases de datos compatibles con Django:

1. SQLite: No se requiere una instalación adicional, ya que SQLite es compatible de forma nativa en Django. Solo necesitas asegurarte de tener el paquete **sqlite3** instalado en tu sistema.
2. PostgreSQL: Para utilizar PostgreSQL como base de datos en Django, debes instalar el paquete **psycopg2**. Puedes instalarlo mediante pip ejecutando el siguiente comando:

    pip install psycopg2
3. MySQL: Si deseas utilizar MySQL como base de datos en Django, debes instalar el paquete **mysqlclient**. Puedes instalarlo mediante pip ejecutando el siguiente comando:

    pip install mysqlclient

    Ten en cuenta que también puedes utilizar el paquete **PyMySQL** como alternativa para conectarte a una base de datos MySQL.
4. Oracle: Para utilizar Oracle como base de datos en Django, debes instalar el paquete **cx_Oracle**. Sin embargo, ten en cuenta que el soporte de Django para Oracle puede requerir configuraciones adicionales y dependencias específicas del sistema. Puedes encontrar más información en la documentación oficial de Django sobre el soporte de Oracle.

La importancia de instalar los paquetes de base de datos correctos en Django radica en que permiten a Django interactuar y comunicarse adecuadamente con la base de datos seleccionada. Los paquetes proporcionan los controladores y las interfaces necesarias para que Django pueda crear, modificar y consultar la estructura y los datos de la base de datos. Sin los paquetes de base de datos instalados, Django no podrá establecer una conexión exitosa y realizar operaciones en la base de datos subyacente.

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

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

1. Abstracción de base de datos: Django proporciona una capa de abstracción de base de datos que permite interactuar con diferentes bases de datos relacionales sin tener que preocuparse por detalles específicos de la base de datos subyacente. Esto facilita el desarrollo y la portabilidad de la aplicación, ya que el código ORM es independiente de la base de datos utilizada.

2. Sintaxis orientada a objetos: Django utiliza una sintaxis orientada a objetos para realizar consultas y manipulaciones de datos en la base de datos. Esto permite escribir consultas de manera más legible y natural, utilizando métodos y atributos de los modelos de Django en lugar de tener que escribir consultas SQL directamente.

3. Mapeo automático de objetos a tablas: Django mapea automáticamente las clases de modelos a tablas de base de datos y las instancias de modelos a filas en esas tablas. Esto simplifica la creación, actualización y eliminación de registros en la base de datos, ya que se pueden realizar estas operaciones a través de los métodos y atributos de los modelos de Django.

4. Gestión automática de relaciones: Django administra automáticamente las relaciones entre los modelos y las tablas de la base de datos, incluyendo las relaciones uno a uno, uno a muchos y muchos a muchos. Esto facilita el manejo de relaciones complejas en la base de datos y simplifica las consultas y operaciones relacionadas.

5. Migraciones automáticas: Django ofrece migraciones automáticas, lo que significa que los cambios en los modelos se traducen en migraciones que describen los cambios en la estructura de la base de datos. Las migraciones se pueden generar y aplicar de manera controlada, evitando la necesidad de escribir y ejecutar manualmente los scripts de cambio de esquema de la base de datos.

6. Seguridad y prevención de inyecciones SQL: El uso de un ORM como Django ayuda a prevenir las vulnerabilidades de seguridad relacionadas con la inyección SQL. Django se encarga de manejar los parámetros de consulta de forma segura, evitando la necesidad de concatenar directamente valores en las consultas SQL.

Estas ventajas hacen que Django como ORM sea una opción poderosa para la integración con una base de datos, ya que simplifica el desarrollo, mejora la legibilidad del código y proporciona una capa de abstracción para trabajar con bases de datos relacionales.