Contenidos a Trabajar

1. Bases de Datos
   - Objetivos de las bases de datos
   - Notas Generales
   - Codificación
   - PostgreSQL
2. ORM
3. Migraciones
   - Los comandos
   - Soporte de back-end

Base de Datos

Objetivo de las bases de datos

Django soporta oficialmente las siguientes bases de datos:

- postgresql
- MariaDB
- mysql
- Oracle
- SQLite

También hay varios backends de bases de datos proporcionados por terceros.

Django intenta admitir tantas funciones como sea posible en todos los backends de bases de datos. Sin embargo, no todos los backends de bases de datos son iguales, y hemos tenido que tomar decisiones de diseño sobre qué características admitir y qué suposiciones podemos hacer de manera segura.

Este archivo describe algunas de las características que pueden ser relevantes para el uso de Django. No pretende reemplazar la documentación específica del servidor ni los manuales de referencia.

Notas generales

Conexiones persistentes

Las conexiones persistentes evitan la sobrecarga de restablecer una conexión a la base de datos en cada solicitud. Están controlados por el parámetro **CONN_MAX_AGE** que define la duración máxima de una conexión. Se puede configurar de forma independiente para cada base de datos.

El valor predeterminado es **0**, conservando el comportamiento histórico de cerrar la conexión de la base de datos al final de cada solicitud. Para habilitar conexiones persistentes, **CONN_MAX_AGE** configúrelo en un número entero positivo de segundos. Para conexiones persistentes ilimitadas, configúrelo en **None**.

Gestión de conexiones

Django abre una conexión a la base de datos cuando hace una consulta a la base de datos por primera

vez. Mantiene esta conexión abierta y la reutiliza en solicitudes posteriores. Django cierra la conexión una vez que excede la edad máxima definida por **CONN_MAX_AGE** o cuando ya no se puede usar.

En detalle, Django abre automáticamente una conexión a la base de datos cada vez que la necesita y aún no la tiene, ya sea porque esta es la primera conexión o porque la conexión anterior se cerró.

Al comienzo de cada solicitud, Django cierra la conexión si ha alcanzado su edad máxima. Si su base de datos finaliza las conexiones inactivas después de un tiempo, debe establecer **CONN_MAX_AGE** a un valor más bajo, de modo que Django no intente utilizar una conexión que haya sido finalizada por el servidor de la base de datos. (Es posible que este problema solo afecte a sitios con muy poco tráfico).

Al final de cada solicitud, Django cierra la conexión si ha alcanzado su antigüedad máxima o si se encuentra en un estado de error irrecuperable. Si se han producido errores en la base de datos durante el procesamiento de las solicitudes, Django comprueba si la conexión aún funciona y la cierra si no es así. Por lo tanto, los errores de la base de datos afectan como máximo a una solicitud; si la conexión se vuelve inutilizable, la siguiente solicitud obtiene una nueva conexión.

Codificación

Django asume que todas las bases de datos utilizan la codificación UTF-8. El uso de otras codificaciones puede generar un comportamiento inesperado, como errores de "valor demasiado largo" de su base de datos para datos que son válidos en Django. Consulte las notas específicas de la base de datos a continuación para obtener información sobre cómo configurar su base de datos correctamente

PostgreSQL

Django es compatible con PostgreSQL 9.6 y superior. Se requiere psycopg2 2.5.4 o superior, aunque serecomienda la última versión.

Configuración de conexión de PostgreSQL

En el archivo de settings.py del proyecto debe configurar la propiedad **DATABASES,** que es un diccionario que contiene la configuración de todas las bases de datos que se utilizarán con Django. Es un diccionario anidado cuyo contenido asigna un alias de base de datos a un diccionario que contiene las opciones para una base de datos individual.

El archivo de configuración más simple posible es para una configuración de base de datos única usandoSQLite. Esto se puede configurar usando lo siguiente:

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

Por ejemplo para PostgreSQL, se muestra la siguiente configuración:

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

Se explican a continuación los parámetros principales:

ENGINE

Predeterminado: ''(Cadena vacía)

El backend de la base de datos a utilizar. Los backends de base de datos incorporados son:

- **'django.db.backends.postgresql'**
- **'django.db.backends.mysql'**
- **'django.db.backends.sqlite3'**
- **'django.db.backends.oracle'**

Puede usar un backend de base de datos que no se envía con Django configurando en **ENGINE** una ruta completamente calificada (es decir, **mypackage.backends.whatever**).

HOST

Predeterminado: **''** (Cadena vacía)

Qué host utilizar al conectarse a la base de datos. Una cadena vacía significa localhost. No se usa con SQLite.

NAME

Predeterminado: **''** (Cadena vacía)

El nombre de la base de datos a utilizar. Para SQLite, es la ruta completa al archivo de la base de datos. Al especificar la ruta, utilice siempre barras diagonales, incluso en Windows

(p.ej: **C:/homes/user/mysite/sqlite3.db**.,).

PORT

Predeterminado: **''** (Cadena vacía)

El puerto a usar cuando se conecta a la base de datos. Una cadena vacía significa el puerto predeterminado. No se usa con SQLite.

USER

Predeterminado: **''** (Cadena vacía)

El nombre de usuario que se utilizará al conectarse a la base de datos. No se usa con SQLite.

PASSWORD

Predeterminado: ''(Cadena vacía)

La contraseña que se utilizará al conectarse a la base de datos. No se usa con SQLite.

Para encontrar más configuraciones visite la documentación oficial: https://docs.djangoproject.com/en/3.2/ref/settings/

ORM

¿Qué es Django ORM?

Django ORM proporciona una forma elegante y poderosa de interactuar con la base de datos. ORM son las siglas de Object Relational Mapping. Es solo una palabra elegante que describe cómo acceder a los datos almacenados en la base de datos de manera orientada a objetos.

¿Cuál es el uso de ORM en Django??

Una de las características más poderosas de Django es su Object-Relational Mapper (ORM), que le permite interactuar con su base de datos, como lo haría con SQL. De hecho, el ORM de Django es solo una forma pitónica de crear SQL para consultar y manipular su base de datos y obtener resultados de una manera pitónica.

¿Qué es ORM en Python??

Un mapeador relacional de objetos (ORM) es una biblioteca de código que automatiza la transferencia de datos almacenados en tablas de bases de datos relacionales a objetos que se usan más comúnmente en elcódigo de la aplicación.

¿Qué es exactamente ORM??

El mapeo relacional de objetos (ORM, O / RM y herramienta de mapeo O / R) en ciencias de la computación es una técnica de programación para convertir datos entre sistemas de tipos incompatibles usando lenguajes de programación orientados a objetos... En la programación orientada a objetos, las tareas de gestión de datos actúan sobre objetos que casi siempre son valores no escalares.

¿Django es compatible con ORM??

El marco web de Django incluye una capa de mapeo relacional de objetos (ORM) predeterminada que se puede usar para interactuar con los datos de la aplicación de varias bases de datos relacionales como SQLite,PostgreSQL y MySQL.

¿Es ORM mejor que SQL??

ORM y SQL son dos herramientas disponibles que los desarrolladores web pueden utilizar en la gestión de bases de datos. Al compararlos, SQL tiene una gestión práctica más alta que ORM. Debido a que ORM tiene un mayor nivel de abstracción y más complejidad que SQL, se requiere menos administración práctica; esto hace que la gestión de datos sea más eficiente.

Migraciones

Las migraciones son la forma en que Django propaga los cambios que realiza en sus modelos (agregar un campo, eliminar un modelo, etc.) en el esquema de su base de datos. Están diseñados para ser en su mayoría automáticos, pero necesitará saber cuándo realizar migraciones, cuándo ejecutarlas y los problemas comunescon los que se puede encontrar.

Los Comandos

Hay varios comandos que utilizará para interactuar con las migraciones y el manejo del esquema de base dedatos de Django:

- **migrate**, que se encarga de aplicar y desaplicar migraciones.
- **makemigrations**, que se encarga de crear nuevas migraciones en función de los cambios que hayarealizado en sus modelos.
- **sqlmigrate**, que muestra las instrucciones SQL para una migración.
- **showmigrations**, que enumera las migraciones de un proyecto y su estado.

Debe pensar en las migraciones como un sistema de control de versiones para el esquema de su base de datos. **Makemigrations** es responsable de empaquetar los cambios de su modelo en archivos de migración individuales, de forma análoga a las confirmaciones, y **migrate** es responsable de aplicarlos a su base de datos.

Los archivos de migración para cada aplicación se encuentran en un directorio de "migraciones" dentro de esa aplicación y están diseñados para comprometerse y distribuirse como parte de su base de código. Debería hacerlos una vez en su máquina de desarrollo y luego ejecutar las mismas migraciones en las máquinas de sus colegas, sus máquinas de ensayo y, finalmente, sus máquinas de producción.

Las migraciones se ejecutarán de la misma manera en el mismo conjunto de datos y producirán resultados consistentes, lo que significa que lo que ve en el desarrollo y la preparación es, bajo las mismas circunstancias, exactamente lo que sucederá en la producción.

Django realizará migraciones para cualquier cambio en sus modelos o campos, incluso las opciones que no afectan la base de datos, ya que la única forma en que

puede reconstruir un campo correctamente es tener todos los cambios en el historial, y es posible que necesite esas opciones en algunas migraciones de datos másadelante (por ejemplo, si ha configurado validadores personalizados).

Soporte de back-end

Las migraciones son compatibles con todos los backends con los que se envía Django, así como con backends de terceros si se han programado para admitir la alteración del esquema (realizado a través de la clase SchemaEditor).

Sin embargo, algunas bases de datos son más capaces que otras cuando se trata de migraciones de esquemas; algunas de las advertencias se tratan a continuación.

PostgreSQL

PostgreSQL es la más capaz de todas las bases de datos aquí en términos de compatibilidad con esquemas.

La única advertencia es que antes de PostgreSQL 11, agregar columnas con valores predeterminados provoca una reescritura completa de la tabla, por un tiempo proporcional a su tamaño. Por esta razón, se recomienda crear siempre nuevas columnas con **null=True**, ya que de esta forma se agregarán de inmediato.

Inicializar proyecto DJANGO

La primera vez que cree un proyecto de Django con su aplicación, y conecte dicho proyecto a una base datos, antes de comenzar a desarrollar los modelos es conveniente aplicar las migraciones por defecto de Django para asegurarse de que se encuentra todo bien configurado y que se generen los datos por defecto de Django para el manejo de migraciones y las aplicaciones instaladas por defecto, como pueden ser el admin de Django, autenticación, etc. Para aplicar estas migraciones debe posicionarse en la carpeta de su proyecto, con el ambiente activado y ejecutar: python manage.py migrate. Dicha ejecución dará un resultado como el que se muestra a continuación si se tienen las mismas aplicaciones instaladas:

```
Operations to perform:
 Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
 Applying contenttypes.0001_initial... OK
 Applying auth.0001_initial... OK
 Applying admin.0001_initial... OK
 Applying admin.0002_logentry_remove_auto_add... OK
 Applying admin.0003_logentry_add_action_flag_choices... OK
 Applying contenttypes.0002_remove_content_type_name... OK
 Applying auth.0002_alter_permission_name_max_length... OK
 Applying auth.0003_alter_user_email_max_length... OK
 Applying auth.0004_alter_user_username_opts... OK
 Applying auth.0005_alter_user_last_login_null... OK
 Applying auth.0006_require_contenttypes_0002... OK
 Applying auth.0007_alter_validators_add_error_messages... OK
 Applying auth.0008_alter_user_username_max_length... OK
 Applying auth.0009_alter_user_last_name_max_length... OK
 Applying auth.0010_alter_group_name_max_length... OK
 Applying auth.0011_update_proxy_permissions... OK
 Applying auth.0012_alter_user_first_name_max_length... OK
 Applying sessions.0001_initial... OK
```