

Conexión de Django con MySQL

Bienvenidos a esta sesión práctica donde aprenderás a integrar Django con MySQL, una de las bases de datos más utilizadas en entornos profesionales. Esta conexión es fundamental para desarrollar aplicaciones web robustas y escalables.

Objetivos de aprendizaje



Conectar Django con MySQL

Dominarás el proceso completo de integración entre el framework y la base de datos



Configurar mysqlclient

Instalarás y configurarás el paquete necesario para la comunicación



Ajustar settings.py

Configurarás correctamente los parámetros de conexión en Django



Ejecutar migraciones

Aprenderás a sincronizar modelos con la base de datos



Metodología activa: Esta clase es 100% práctica. Aprenderás haciendo cada paso dentro de VS Code y MySQL. Prepárate para experimentar, probar y construir tu propio proyecto Bikeshop desde cero.

¿Por qué Django necesita una base de datos?

Django actúa como un puente inteligente entre tu código Python y la información que necesitas almacenar. Sin una base de datos, tus aplicaciones web no podrían guardar ni recuperar información de forma estructurada y persistente.

La base de datos es el corazón de cualquier aplicación web moderna. Es donde vive toda la información crítica de tu proyecto.



Tienda online



Guarda catálogos de bicicletas, precios, inventario y pedidos de clientes

Blog personal

Almacena artículos, comentarios de usuarios y categorías de contenido

App escolar

Registra estudiantes, calificaciones, asistencias y materias

  **Actividad inicial:** Piensa y comenta con tu compañero: ¿Qué tipo de datos debería guardar la aplicación web que estás desarrollando en el bootcamp? ¿Usuarios? ¿Productos? ¿Transacciones?

Motores de bases de datos compatibles con Django

Django es extremadamente flexible y puede conectarse con múltiples sistemas de gestión de bases de datos. Cada uno tiene sus propias ventajas según el tipo de proyecto que estés desarrollando.



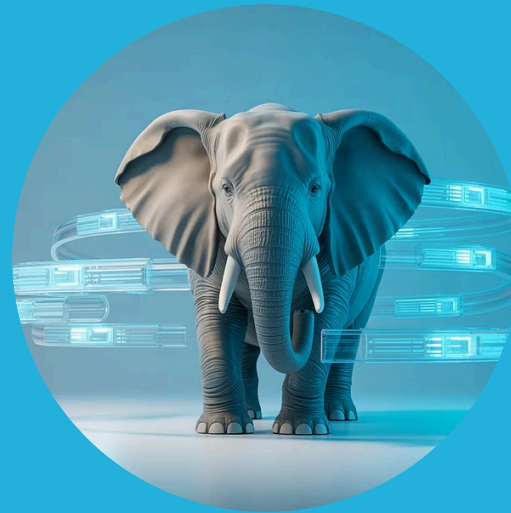
SQLite

Base de datos por defecto en Django. Perfecta para desarrollo y pruebas rápidas. No requiere servidor.



MySQL

La más popular en producción. Rápida, confiable y ampliamente soportada en la industria.



PostgreSQL

Potente y con características avanzadas. Ideal para aplicaciones empresariales complejas.



Oracle

Solución empresarial robusta. Usada en grandes corporaciones con altos volúmenes de datos.

¿Por qué elegimos MySQL para esta clase?

- Rendimiento excepcional: Optimizado para operaciones rápidas de lectura y escritura
- Escalabilidad probada: Soporta desde pequeños proyectos hasta aplicaciones masivas
- Amplia adopción: Usado por gigantes como Facebook, Twitter y YouTube
- Documentación extensa: Gran comunidad y recursos de aprendizaje disponibles
- Gestión multiusuario: Permite trabajo colaborativo con permisos granulares



El poder del ORM de Django

El ORM (Object-Relational Mapping) es una de las características más poderosas de Django. Actúa como un traductor mágico entre tu código Python orientado a objetos y el lenguaje SQL que entiende la base de datos.

Código Python (Django)

```
Bicicleta.objects.create(  
    marca="Trek",  
    modelo="Marlin 7",  
    precio=15999.00  
)
```

Escribes código Python natural y orientado a objetos

SQL generado automáticamente

```
INSERT INTO bicicletas_bicicleta  
(marca, modelo, precio)  
VALUES  
( 'Trek', 'Marlin 7', 15999.00);
```

Django lo convierte en SQL válido detrás de escena

✓ No escribes SQL manualmente

Olvídate de la sintaxis compleja de SQL. Django se encarga de todo.

✓ Trabajas con objetos Python

Usa clases, métodos y atributos como en cualquier programa Python.

✓ Portabilidad de código

Cambia de MySQL a PostgreSQL sin reescribir tu código.

✓ Prevención de inyección SQL

El ORM sanitiza automáticamente tus consultas por seguridad.

Instalación de mysqlclient

Para que Django pueda comunicarse con MySQL, necesitamos instalar un driver de conexión llamado `mysqlclient`. Este paquete actúa como el intérprete que permite que ambas tecnologías se entiendan perfectamente.

01

Activa tu entorno virtual

Asegúrate de estar dentro de tu entorno virtual de Python antes de instalar paquetes

02

Ejecuta el comando de instalación

```
pip install mysqlclient
```

03

Verifica la instalación exitosa

Busca el mensaje "Successfully installed mysqlclient" en tu terminal

Solución de problemas en Windows

Si encuentras errores de compilación en Windows, tienes dos opciones confiables:

- Opción recomendada: Instalar Anaconda, que incluye compiladores preconfigurados
- Alternativa: Descargar e instalar Visual C++ Build Tools desde Microsoft

 **Actividad práctica:** Instala `mysqlclient` en tu entorno virtual ahora mismo. Comparte en el chat si tu instalación fue exitosa o si necesitas ayuda con algún error.

Creación del proyecto Django

Comenzaremos creando la estructura base de nuestro proyecto. Django organiza el código en proyectos (contenedores principales) y aplicaciones (módulos funcionales específicos).

Comandos de creación

```
django-admin startproject bikeshop_project
```

```
cd bikeshop_project
```

Estos comandos generan toda la estructura inicial necesaria para tu proyecto Django.

Archivos clave generados:

- `settings.py` - Configuración del proyecto
- `urls.py` - Rutas de la aplicación
- `manage.py` - Utilidad de línea de comandos
- `wsgi.py` - Punto de entrada del servidor



 Punto importante: El archivo `settings.py` será donde configuraremos toda la conexión con MySQL en los próximos pasos.

1

bikeshop_project/
Directorio raíz del proyecto que contiene todo

2

bikeshop_project/bikeshop_project/
Paquete Python con archivos de configuración

3

manage.py
Script para ejecutar comandos administrativos

Creación de la aplicación Bicicletas

En Django, un proyecto puede contener múltiples aplicaciones. Cada aplicación es un módulo independiente con una funcionalidad específica. Vamos a crear nuestra primera app llamada "bicicletas" que manejará todo lo relacionado con nuestro catálogo de bicicletas.

Comando de creación

```
python manage.py startapp bicicletas
```

Estructura generada

```
bicicletas/  
├── migrations/  
├── __init__.py  
├── admin.py  
├── apps.py  
├── models.py  
├── tests.py  
└── views.py
```

Archivos principales

- **models.py:** Define la estructura de tus datos (tablas)
- **views.py:** Contiene la lógica de negocio
- **admin.py:** Registra modelos en el panel administrativo
- **migrations/:** Historial de cambios en la base de datos

Paso crucial: Registrar la aplicación

Después de crear la app, debes agregarla a **INSTALLED_APPS** en **settings.py**:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'bicicletas', # ← Agrega esta línea  
]
```

 **Actividad:** Crea la app "bicicletas" y regístrala en **INSTALLED_APPS**. Verifica que Django la reconozca ejecutando `python manage.py check`

Creación de la base de datos en MySQL

Antes de que Django pueda conectarse, necesitamos crear la base de datos en el servidor MySQL. Este es un paso manual que solo haremos una vez.



Buenas prácticas de nomenclatura

- Usa nombres descriptivos y en minúsculas
- Evita espacios (usa guiones bajos si necesitas separar palabras)
- Relaciona el nombre con el propósito del proyecto
- Mantén consistencia entre el nombre del proyecto Django y la base de datos









💡 **Consejo profesional:** Si prefieres trabajar con interfaz gráfica, MySQL Workbench te permite crear bases de datos con clics. Es especialmente útil cuando estás aprendiendo o necesitas visualizar la estructura de tus tablas.



Configuración de la conexión en settings.py

Ahora viene la parte crucial: indicarle a Django dónde está nuestra base de datos MySQL y cómo conectarse a ella. Toda esta configuración se realiza en el diccionario `DATABASES` dentro de `settings.py`.

Configuración completa

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'bikeshop',
        'USER': 'root',
        'PASSWORD': 'tu_contraseña_mysql',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

	ENGINE Especifica qué motor de base de datos usar. Para MySQL siempre será 'django.db.backends.mysql'
	NAME El nombre exacto de la base de datos que creaste en MySQL (en nuestro caso: 'bikeshop')
	USER Usuario de MySQL con permisos para acceder a la base de datos (comúnmente 'root' en desarrollo)
	PASSWORD La contraseña de tu usuario MySQL. ¡Nunca compartas este archivo con la contraseña real en producción!
	HOST Dirección del servidor MySQL. 'localhost' o '127.0.0.1' para desarrollo local
	PORT Puerto en el que MySQL escucha conexiones. El puerto por defecto es '3306'

 **Importante: Seguridad en producción**

Nunca subas contraseñas reales a repositorios públicos. En producción, usa variables de entorno o servicios de gestión de secretos para manejar credenciales sensibles.

Primera verificación de conexión

Es el momento de la verdad: vamos a probar si Django puede comunicarse correctamente con MySQL. Usaremos el comando `migrate` que, además de probar la conexión, creará las tablas internas que Django necesita.

Comando de verificación

```
python manage.py migrate
```

Este comando hace dos cosas:

- 1. Verifica que la conexión a MySQL funcione
- 2. Crea las tablas necesarias para el sistema de autenticación, sesiones, etc.



Salida esperada (conexión exitosa)

```
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 sessions.0001_initial... OK
```

☒ **auth.0001_initial**

Sistema de usuarios y permisos

☒ **contenttypes.0001_initial**

Registro de tipos de contenido

☒ **sessions.0001_initial**

Manejo de sesiones de usuario

☒ **admin.0001_initial**

Panel de administración

🎉 ¡Felicitaciones! Si ves estos mensajes, significa que Django se conectó exitosamente a MySQL y creó todas las tablas necesarias. Tu entorno está listo para comenzar a trabajar.

Actividad práctica: Simulación de error

Los errores son parte natural del desarrollo. Aprender a interpretarlos y solucionarlos es una habilidad fundamental. Vamos a provocar intencionalmente un error de conexión para que sepas cómo identificarlo y corregirlo.

01

Modifica el nombre de la base de datos

En settings.py, cambia 'bikeshop' por 'bikeshop_falso' en la clave NAME

02

Intenta ejecutar migrate nuevamente

```
python manage.py migrate
```

03

Observa y analiza el mensaje de error

Django te mostrará un error específico indicando que no puede conectarse

04


Corrige el error

Vuelve a cambiar el nombre a 'bikeshop' y verifica que funcione

Mensaje de error típico

```
django.db.utils.OperationalError: (1049, "Unknown database 'bikeshop_falso'")
```

Este error te indica claramente que MySQL no encuentra la base de datos especificada.

 **Reflexión grupal:** Discute con tu compañero: ¿Por qué son importantes los mensajes de error? ¿Qué información útil te proporcionó Django en este caso? ¿Cómo puedes usar esta información para resolver problemas futuros?

Creación del modelo Bicicleta

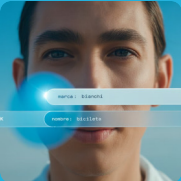
Los modelos son clases Python que representan las tablas de tu base de datos. Cada atributo de la clase se convierte en una columna de la tabla. Django se encarga automáticamente de traducir esta estructura a SQL.

Definición del modelo en bicicletas/models.py

```
from django.db import models

class Bicicleta(models.Model):
    marca = models.CharField(max_length=50)
    modelo = models.CharField(max_length=50)
    tipo = models.CharField(max_length=20)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    disponible = models.BooleanField(default=True)
    anio = models.IntegerField()

    def __str__(self):
        return f'{self.marca} {self.modelo}'
```



CharField
Para texto corto. Requiere max_length. Ideal para nombres, títulos, códigos.



DecimalField
Para números decimales precisos. Perfecto para precios y monedas. Evita errores de redondeo.



BooleanField
Para valores True/False. Útil para estados, flags, disponibilidad.



IntegerField
Para números enteros. Ideal para cantidades, años, identificadores numéricos.

🧩 Resultado en MySQL

Django convertirá esta clase en una tabla llamada:

bicicletas_bicicleta

El nombre sigue el patrón:

nombreapp_nombremodelo

📊 Columnas generadas

- id (clave primaria automática)
- marca (VARCHAR 50)
- modelo (VARCHAR 50)
- tipo (VARCHAR 20)
- precio (DECIMAL 10,2)
- disponible (BOOLEAN)
- anio (INTEGER)

Sistema de migraciones de Django

Las migraciones son el mecanismo de Django para propagar cambios en tus modelos a la estructura de la base de datos. Funcionan como un sistema de control de versiones para tu esquema de base de datos.



Modificas models.py

Defines o cambias tus modelos



makemigrations

Django crea archivos de migración



migrate

Aplica cambios a la base de datos

1 Crear las migraciones

```
python manage.py makemigrations bicicletas
```

Salida esperada:

```
Migrations for 'bicicletas':
  bicicletas/migrations/0001_initial.py
  - Create model Bicicleta
```

Django analiza tu modelo y genera un archivo de migración con las instrucciones SQL necesarias.

2 Aplicar las migraciones

```
python manage.py migrate
```

Salida esperada:

```
Running migrations:
  Applying bicicletas.0001_initial... OK
```

Django ejecuta la migración y crea la tabla en MySQL con todas las columnas definidas.



Concepto clave: Las migraciones son incrementales. Cada cambio que hagas en tus modelos generará una nueva migración. Django mantiene un historial completo de la evolución de tu base de datos, permitiéndote avanzar o retroceder en el tiempo.

Creación del superusuario

Django incluye un poderoso panel de administración que te permite gestionar los datos de tu aplicación sin escribir código. Para acceder a él, primero necesitas crear un usuario administrador.

Comando de creación

```
python manage.py createsuperuser
```

El comando te pedirá:

- Nombre de usuario
- Dirección de correo electrónico
- Contraseña (mínimo 8 caracteres)
- Confirmación de contraseña



Inicia el servidor de desarrollo

```
python manage.py runserver
```



Accede al panel de administración

Abre tu navegador y ve a:
<http://127.0.0.1:8000/admin>



Ingresas tus credenciales
Usa el nombre de usuario y contraseña que acabas de crear

Registrar el modelo en el admin

Para ver el modelo Bicicleta en el panel, edita `bicicletas/admin.py`:

```
from django.contrib import admin
from .models import Bicicleta

admin.site.register(Bicicleta)
```

Ahora podrás ver, crear, editar y eliminar bicicletas desde la interfaz administrativa.

Insertando datos con el ORM

El verdadero poder del ORM se ve cuando empiezas a manipular datos. Django proporciona una API Python elegante y expresiva para crear, leer, actualizar y eliminar registros.

01

Abre el shell interactivo de Django

```
python manage.py shell
```

Este shell es como el intérprete de Python pero con Django configurado

02

Importa tu modelo

```
from bicicletas.models import Bicicleta
```

03

Crea un nuevo registro

```
Bicicleta.objects.create(
    marca="Giant",
    modelo="Talon 1",
    tipo="mtb",
    precio=999.99,
    disponible=True,
    anio=2024
)
```

Código Python (ORM)

```
bike = Bicicleta.objects.create(
    marca="Trek",
    modelo="Marlin 7",
    tipo="mtb",
    precio=15999.00,
    disponible=True,
    anio=2024
)

print(f"Creada: {bike}")
```

SQL equivalente (generado automáticamente)

```
INSERT INTO bicicletas_bicicleta
(marca, modelo, tipo, precio,
disponible, anio)
VALUES
('Trek', 'Marlin 7', 'mtb',
15999.00, 1, 2024);
```

💡 **Ventaja del ORM:** Django convierte automáticamente tu código Python en sentencias SQL válidas, sanitiza los datos para prevenir inyección SQL, y devuelve objetos Python que puedes manipular fácilmente.

Consultas con el ORM: QuerySets

Los QuerySets son el corazón del ORM de Django. Te permiten recuperar datos de la base de datos usando métodos Python encadenables, potentes y expresivos.



Obtener todos los registros

```
Bicicleta.objects.all()
```

Devuelve un QuerySet con todas las bicicletas



Filtrar por condición

```
Bicicleta.objects.filter(
    disponible=True
)
```

Solo bicicletas disponibles



Obtener un solo registro

```
Bicicleta.objects.get(
    id=1
)
```

Lanza excepción si no existe o hay múltiples



Excluir registros

```
Bicicleta.objects.exclude(
    tipo="ruta"
)
```

Todas excepto las de ruta

Consultas avanzadas con múltiples filtros

```
# Bicicletas MTB disponibles del 2024
```

```
mtb_2024 = Bicicleta.objects.filter(
    tipo="mtb",
    disponible=True,
    anio=2024
)
```

```
# Bicicletas con precio menor a 20000
```

```
economicas = Bicicleta.objects.filter(precio__lt=20000)
```

```
# Bicicletas cuya marca comienza con "G"
```

```
marca_g = Bicicleta.objects.filter(marca__startswith="G")
```

```
# Ordenar por precio de mayor a menor
```

```
caras_primero = Bicicleta.objects.all().order_by('-precio')
```



Actividad: Abre el shell de Django y prueba estas consultas: filtra bicicletas por tipo, busca por año de fabricación, ordena por precio. Experimenta con diferentes combinaciones y observa los resultados.

Accediendo a MySQL directamente con dbshell

Aunque el ORM es poderoso, a veces quieres ver exactamente qué hay en la base de datos usando SQL puro. Django proporciona el comando `dbshell` que te conecta directamente al cliente MySQL.

Acceso con dbshell

```
python manage.py dbshell
```

Este comando abre una sesión MySQL directa usando las credenciales de `settings.py`

Consultas SQL directas

```
-- Ver todas las bicicletas
SELECT * FROM bicicletas_bicicleta;

-- Filtrar por marca
SELECT marca, modelo, precio
FROM bicicletas_bicicleta
WHERE marca = 'Giant';

-- Contar registros
SELECT COUNT(*) FROM bicicletas_bicicleta;
```



👁️ Comparación visual: Ejecuta la misma consulta en el ORM y en dbshell. Observa cómo los datos son exactamente los mismos, pero presentados de diferente forma.

ORM: Objetos Python

```
>>> bikes = Bicicleta.objects.all()
>>> for b in bikes:
...     print(b.marca, b.modelo)
```

Trabajas con objetos y atributos

dbshell: Resultados SQL

```
mysql> SELECT marca, modelo
      FROM bicicletas_bicicleta;
```

Ves filas y columnas crudas

💡 **Uso recomendado:** Usa el ORM para el 99% de tus operaciones. Usa dbshell para debugging, consultas de análisis complejas, o cuando necesitas entender exactamente qué está pasando en la base de datos.

Ejercicio colaborativo en parejas

Es hora de poner en práctica todo lo aprendido. Trabajarán en parejas para crear, consultar, modificar y eliminar registros usando el ORM de Django.

1

Crear 3 bicicletas diferentes

Usa `Bicicleta.objects.create()` para agregar:

- Una bicicleta de montaña (MTB)
- Una bicicleta de ruta
- Una bicicleta urbana

Asegúrate de usar datos realistas y variados

2

Listar todas las bicicletas

Usa `Bicicleta.objects.all()` e itera sobre los resultados para imprimir marca y modelo de cada una

3

Modificar el precio de una bicicleta

Recupera una bicicleta específica con `get()`, cambia su precio, y guarda con `.save()`

```
bike = Bicicleta.objects.get(id=1)
bike.precio = 18999.00
bike.save()
```

4

Eliminar una bicicleta

Selecciona una bicicleta y elimínala usando `.delete()`

```
bike = Bicicleta.objects.get(id=2)
bike.delete()
```



Discusión en parejas

Compartan con su compañero:

- ¿Qué línea de código usaste para cada operación?
- ¿Encontraste algún error? ¿Cómo lo resolviste?
- ¿Qué ventajas ves en usar el ORM versus SQL directo?

Diagnóstico y resolución de errores comunes

Los errores son oportunidades de aprendizaje. Conocer los problemas más frecuentes y sus soluciones te ahorrará horas de frustración.

❌

django.db.utils.OperationalError

Mensaje: "Access denied for user 'root'@'localhost'"

Causa: Usuario o contraseña incorrectos en settings.py

Solución: Verifica que USER y PASSWORD en DATABASES coincidan exactamente con tus credenciales de MySQL

❌

ModuleNotFoundError: No module named 'MySQLdb'

Causa: El paquete mysqlclient no está instalado o está en un entorno virtual diferente

Solución: Asegúrate de estar en tu entorno virtual correcto y ejecuta: `pip install mysqlclient`

❌

django.db.utils.OperationalError: (2003)

Mensaje: "Can't connect to MySQL server on 'localhost'"

Causa: El servidor MySQL no está corriendo

Solución: Inicia MySQL con `net start mysql` (Windows) o `sudo service mysql start` (Linux/Mac)

❌

django.db.utils.OperationalError: (1049)

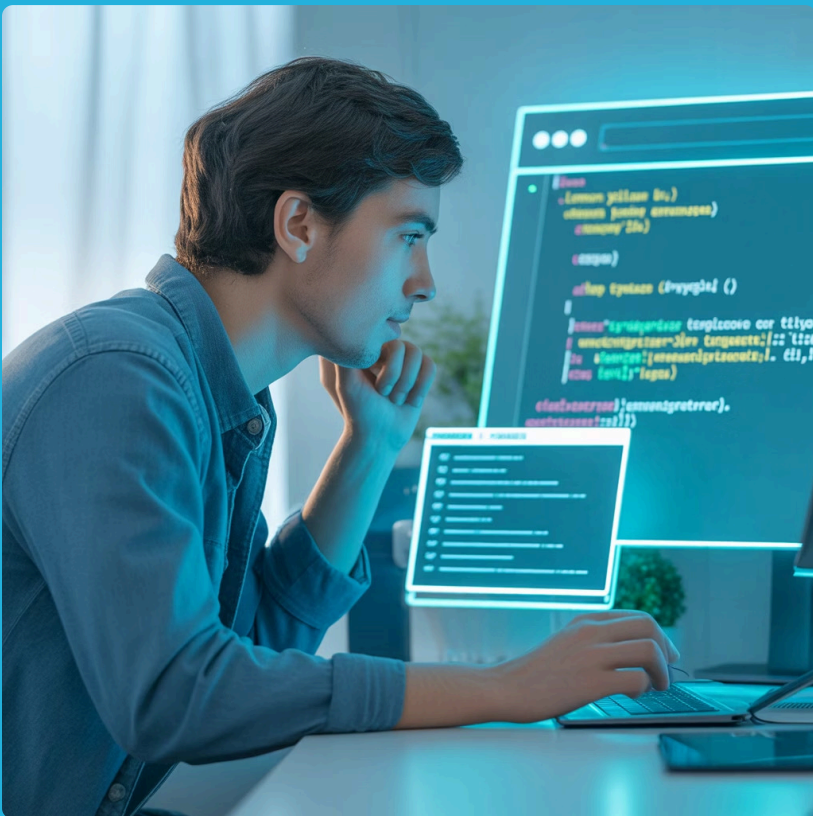
Mensaje: "Unknown database 'bikeshop'"

Causa: La base de datos especificada no existe en MySQL

Solución: Conéctate a MySQL y ejecuta: `CREATE DATABASE bikeshop;`

🔧 Comandos de diagnóstico útiles

- `python manage.py check` - Verifica configuración de Django
- `python manage.py showmigrations` - Muestra estado de migraciones
- `python manage.py dbshell` - Prueba conexión directa a MySQL
- `pip list | grep mysql` - Verifica instalación de mysqlclient



📄

💡 Consejo profesional: Antes de buscar ayuda, ejecuta `python manage.py check`. Este comando detecta la mayoría de problemas de configuración y te da pistas específicas sobre qué revisar.