

Unidad didáctica 02

Aplicaciones con bases de datos relacionales



Índice

UD 02: Aplicaciones con bases de datos relacionales.....	4
1. Ventajas e inconvenientes de utilizar conectores.....	4
Ventajas de los conectores:.....	4
Inconvenientes de los conectores:.....	4
Actividad 1 de clase.....	5
2. Uso de gestores de bases de datos embebidos e independientes.....	6
Ejercicio práctico 1: Crear una base de datos SQLite.....	6
El objeto “cursor”.....	8
Ejercicio práctico 2: Con la base de datos MySQL.....	9
Actividad 2 de clase.....	12
3. Selección del conector idóneo en la aplicación.....	13
¿Cuál escoger?.....	13
Actividad 3 de clase.....	13
4. Definición de la estructura de la base de datos.....	17
Actividad 4 de clase.....	18
5. Modificación del contenido de la base de datos.....	19
Actividad 5 de clase:.....	19
6. Definición de objetos para almacenar resultados de consultas.....	20
Ejercicio Práctico 3: Uso de los métodos fetchone(), fetchall() y fetchmany().....	20
Actividad 6 de clase:.....	22
7. Gestión de transacciones.....	23
Ejercicio Práctico 4: Implementar una transacción:.....	23
Actividad 7 de clase:.....	24
8. Ejecución de procedimientos almacenados.....	25
Ejercicio Práctico 5: Crear y ejecutar un procedimiento almacenado:.....	25
El método stored_results().....	26
Actividad 8 de clase.....	28
Anexo A. Virtualenv.....	29
¿Qué es `virtualenv`?.....	29
¿Para qué sirve `virtualenv`?.....	29
Ventajas de `virtualenv`:.....	29
Ejemplo Práctico: Uso de virtualenv.....	29
¿Qué ocurre si no usamos `virtualenv`?.....	30
¿Qué ocurriría sin `virtualenv`?.....	32
Anexo B. Puesta en marcha del servidor de MySQL en Ubuntu 24.04.....	34
B.1. Cambio de la clave de root.....	35
B.2. Creación de una base de datos y una tabla.....	36
B.3. Creación de usuario no administrador y darle permisos sobre nuestra base de datos.....	36

Anexo C. Configurando nuestro entorno Python.....	38
---	----

UD 02: Aplicaciones con bases de datos relacionales

Resultado de Aprendizaje: Desarrolla aplicaciones que gestionan información almacenada en bases de datos relacionales identificando y utilizando mecanismos de conexión

Para la realización de las actividades propuestas en esta tabla es imprescindible que el alumnado tenga a su disposición una base de datos MySQL local con al menos una tabla. (ver anexos).

También es imprescindible que el alumnado se haya familiarizado con la herramienta pip.

Además es recomendable usar entornos virtuales de Python (ver anexos).

1. Ventajas e inconvenientes de utilizar conectores

Objetivo:

Entender cuándo es adecuado utilizar un conector de base de datos y qué impacto puede tener en el rendimiento, portabilidad y complejidad.

Explicación Teórica:

Un conector de base de datos es un software que actúa como intermediario entre una aplicación y una base de datos, permitiendo a la aplicación realizar operaciones CRUD (Crear, Leer, Actualizar, Borrar). En el caso de MySQL, el conector más utilizado en Python es ``mysql-connector-python``, aunque también existe ``PyMySQL`` y ``SQLAlchemy``.

Ventajas de los conectores:

- Simplicidad en la interacción con la base de datos: Permiten ejecutar consultas SQL directamente desde Python.
- Compatibilidad multiplataforma* Facilitan la ejecución de aplicaciones en diversos sistemas operativos siempre que el conector esté disponible.
- Manejo de errores y excepciones: Proveen mecanismos para capturar y gestionar errores.
- Soporte para transacciones: Facilitan el manejo de transacciones (commit y rollback) de forma nativa.

Inconvenientes de los conectores:

- Rendimiento: En algunos casos, puede haber una pequeña pérdida de rendimiento debido a la abstracción que proporciona el conector.

- Compatibilidad limitada: No todos los conectores soportan todas las funcionalidades avanzadas de MySQL (por ejemplo, algunas extensiones específicas o procedimientos almacenados complejos).
- Dependencia de terceros: El desarrollo del conector depende de su comunidad o empresa, lo que puede generar incertidumbre en cuanto a actualizaciones o corrección de errores.

Ejercicio práctico para hacer en clase:

1. Instalación del conector:

Los alumnos deben instalar el conector de MySQL utilizando el siguiente comando:

```
pip install mysql-connector-python
```

2. Prueba de conexión básica:

Escribimos un programa que se conecte a una base de datos MySQL local, compruebe la conexión y maneje posibles errores.

```
import mysql.connector

from mysql.connector import Error

try:
    conexion = mysql.connector.connect(
        host='localhost',
        user='root',
        password='password',
        database='mi_base_datos'
    )
    if conexion.is_connected():
        print("Conexión a la base de datos exitosa")
except Error as e:
    print(f"Error de conexión: {e}")
finally:
    if conexion.is_connected():
        conexion.close()
    print("Conexión cerrada")
```

Actividad 1 de clase.

Haz la misma actividad anterior, pero con otro conector, denominado “PyMySQL”. Debes abrir la conexión a base de datos, cerrarla en caso de éxito y gestionar los errores. Además, en caso de error, también debe cerrar la conexión con la base de datos.

Adjunta captura de pantalla de Visual Studio Code en el que se vea el código y la traza de ejecución del programa en el terminal incrustado de Code.

2. Uso de gestores de bases de datos embebidos e independientes

El objetivo de este apartado es entender las diferencias entre bases de datos embebidas e independientes, y saber cuándo usar una u otra según los requisitos de la aplicación.

Las Bases de datos embebidas como **SQLite** están integradas en la aplicación, lo que significa que no requieren un servidor separado. Son ideales para aplicaciones pequeñas o donde la simplicidad y la portabilidad son importantes. SQLite almacena los datos en un solo archivo, lo que lo hace fácil de transportar.

Las bases de datos independientes como **MySQL** o **PostgreSQL** requieren un servidor independiente que maneje las conexiones. Son adecuadas para aplicaciones más grandes con múltiples usuarios, ya que pueden manejar mayores volúmenes de datos, concurrencia y seguridad.

Comparativa:

Característica	SQLite	MySQL
Tipo de Base de Datos	Embebida	Independiente
Instalación	No requiere instalación	Requiere servidor MySQL
Rendimiento	Bueno para aplicaciones pequeñas	Alto rendimiento en aplicaciones grandes
Concurrencia	Limitada	Muy alta
Complejidad	Baja	Media-alta

Ejercicio práctico 1: Crear una base de datos SQLite.

Crearemos un archivo SQLite, agregar datos y realizar operaciones CRUD utilizando Python. Esto les ayudará a entender cómo funciona una base de datos embebida.

```
import sqlite3

conexion = sqlite3.connect('mi_base_datos.db')

cursor = conexion.cursor()

#Se usa las comillas triples para poder fragmentar las líneas en varias partes

cursor.execute("""
    CREATE TABLE IF NOT EXISTS alumnos (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nombre TEXT,
        edad INTEGER
    )
""")

cursor.execute(
    "INSERT INTO alumnos (nombre, edad) VALUES (?, ?)", ("Juan", 20))

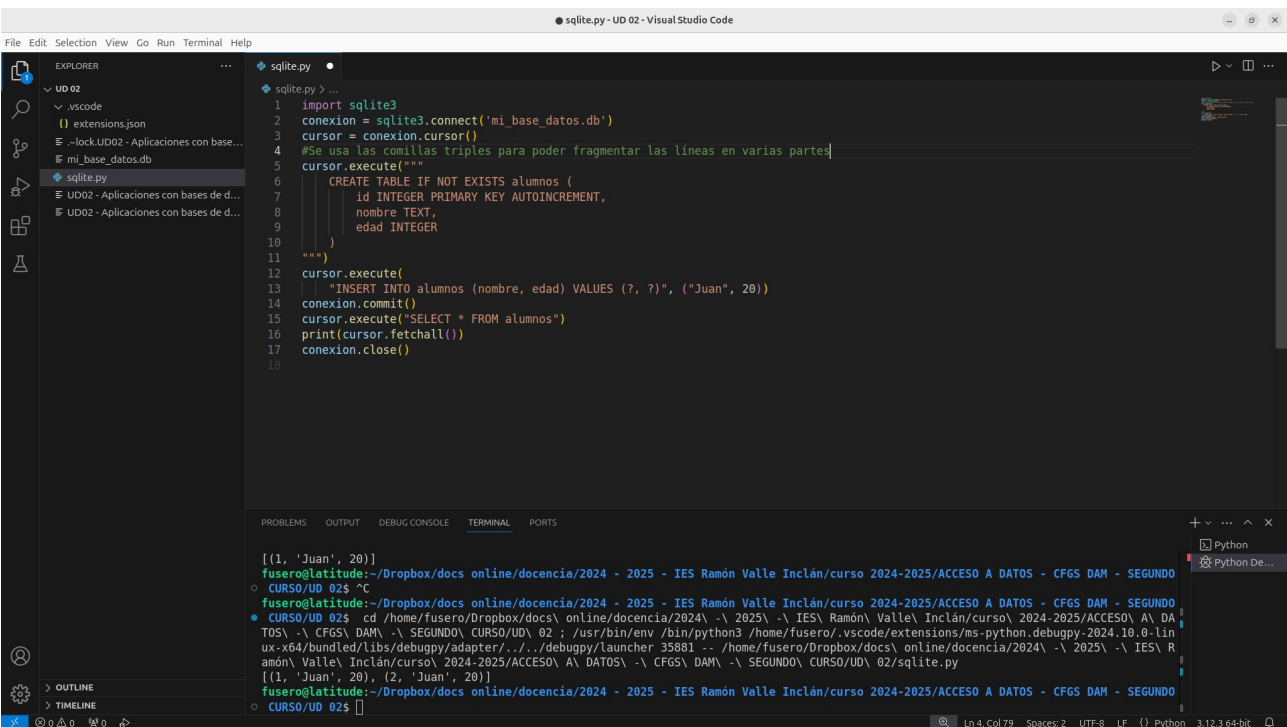
conexion.commit()

cursor.execute("SELECT * FROM alumnos")

print(cursor.fetchall())

conexion.close()
```

Veamos la traza de la ejecución con Visual Studio Code:



Veamos también como ha quedado la base de datos SQLite después de la ejecución del script de Python, llamando a la base de datos desde la línea de comandos:

```
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open mi_base_datos.db
sqlite> select * from alumnos;
1|Juan|20
sqlite>
```

El objeto “cursor”

En este ejemplo, se puede observar que se está usando el objeto **cursor**. Éste es una herramienta fundamental cuando trabajas con bases de datos en Python, específicamente con SQLite (pero el concepto es aplicable a otras bases de datos).

El cursor es un objeto que actúa como un intermediario entre la aplicación (en este caso, tu programa en Python) y la base de datos. Cuando se realiza una consulta SQL (como SELECT, INSERT, UPDATE, etc.), el cursor ejecuta esa consulta y gestiona los resultados.

Veamos lo que hace nuestro código de ejemplo:

Primero, estableces una conexión a la base de datos con `sqlite3.connect()`, y luego creas el cursor utilizando `conexion.cursor()`. A partir de este momento, el cursor es responsable de ejecutar consultas y obtener resultados.

- **Crear una tabla:**

```
cursor.execute("CREATE TABLE IF NOT EXISTS alumnos (id INTEGER PRIMARY KEY AUTOINCREMENT, nombre TEXT, edad INTEGER)")
```

Aquí el cursor ejecuta una consulta SQL que crea una tabla llamada `alumnos` si no existe ya. El método `execute()` se encarga de enviar esta consulta a la base de datos.

- **Insertar datos:**

```
cursor.execute("INSERT INTO alumnos (nombre, edad) VALUES (?, ?)", ("Juan", 20))
```

Con esta línea, el cursor ejecuta una consulta de inserción, donde se añaden los valores "Juan" y 20 en las columnas nombre y edad. Los signos de interrogación (?) son marcadores de posición que son reemplazados por los valores del segundo argumento de `execute()`. Este mecanismo evita inyecciones SQL.

A modo aclaratorio, la línea INSERT que no tiene en cuenta los ataques por inyecciones SQL es ésta:

```
cursor.execute(f"INSERT INTO alumnos (nombre, edad) VALUES ('Juan', 20)")
```


Guardar los cambios:

```
conexion.commit()
```

Después de insertar los datos, se usa `commit()` para confirmar los cambios en la base de datos. Esto es necesario porque, sin ello, los cambios no se guardarían de forma permanente. Aunque la consulta `INSERT INTO` se ejecuta cuando llamas a `execute()`, los cambios no se guardan de forma permanente en la base de datos hasta que llamas a `commit()`. Esto es importante en las bases de datos que usan **transacciones**, como SQLite. **Hasta que no se llama a `commit()`, los cambios pueden ser revertidos.**

Ejercicio práctico 2: Con la base de datos MySQL

Para convertir el programa de SQLite a MySQL, debes realizar algunos ajustes, principalmente en la forma en que te conectas a la base de datos y cómo utilizas el conector de MySQL. En este caso, utilizaremos la librería `mysql-connector-python`, que es el conector oficial de MySQL para Python.

Antes de empezar, asegúrate de tener el conector de MySQL instalado en tu entorno virtual.

Creación y activación de entorno virtual

```
mkdir ejemplo_practico2_mysql
cd ejemplo_practico2_mysql
virtualenv venv_ejemplo_practico2_mysql
source venv_ejemplo_practico2_mysql/bin/activate
```

Instalación del conector de MySQL

```
pip install mysql-connector-python
```

Abrimos Visual Studio Code

```
code .
```

Usamos este código (adaptado para MySQL en lugar de SQLite)

```
import mysql.connector
# Conexión a la base de datos MySQL
conexion = mysql.connector.connect(
    host="localhost",    # Cambia esto si tu base de datos está en otro servidor
    user="usuario",     # Tu usuario de MySQL
    password="usuario", # Tu contraseña de MySQL
    database="ldam"     # Asegúrate de que la base de datos exista
)
cursor = conexion.cursor()
# Crear la tabla si no existe
```

```
cursor.execute("""
    CREATE TABLE IF NOT EXISTS alumnos (
        id INT AUTO_INCREMENT PRIMARY KEY,
        nombre VARCHAR(255),
        edad INT
    )
""")
# Insertar datos en la tabla
cursor.execute(
    "INSERT INTO alumnos (nombre, edad) VALUES (%s, %s)",
    ("Juan", 20)
)
# Confirmar los cambios
conexion.commit()
# Realizar una consulta para recuperar los datos
cursor.execute("SELECT * FROM alumnos")
# Imprimir todos los registros
for fila in cursor.fetchall():
    print(fila)
# Cerrar la conexión
conexion.close()
```

Crear la base de datos en MySQL

Antes de ejecutar este script, debes asegurarte de que la base de datos `1dam` ya exista en tu servidor MySQL. Si no existe, consulta el Anexo B.

Ejecuta tu programa en Python

Luego, ejecuta tu programa en Python (Ctrl+F5) y debería conectarse a la base de datos MySQL sin problemas.¹

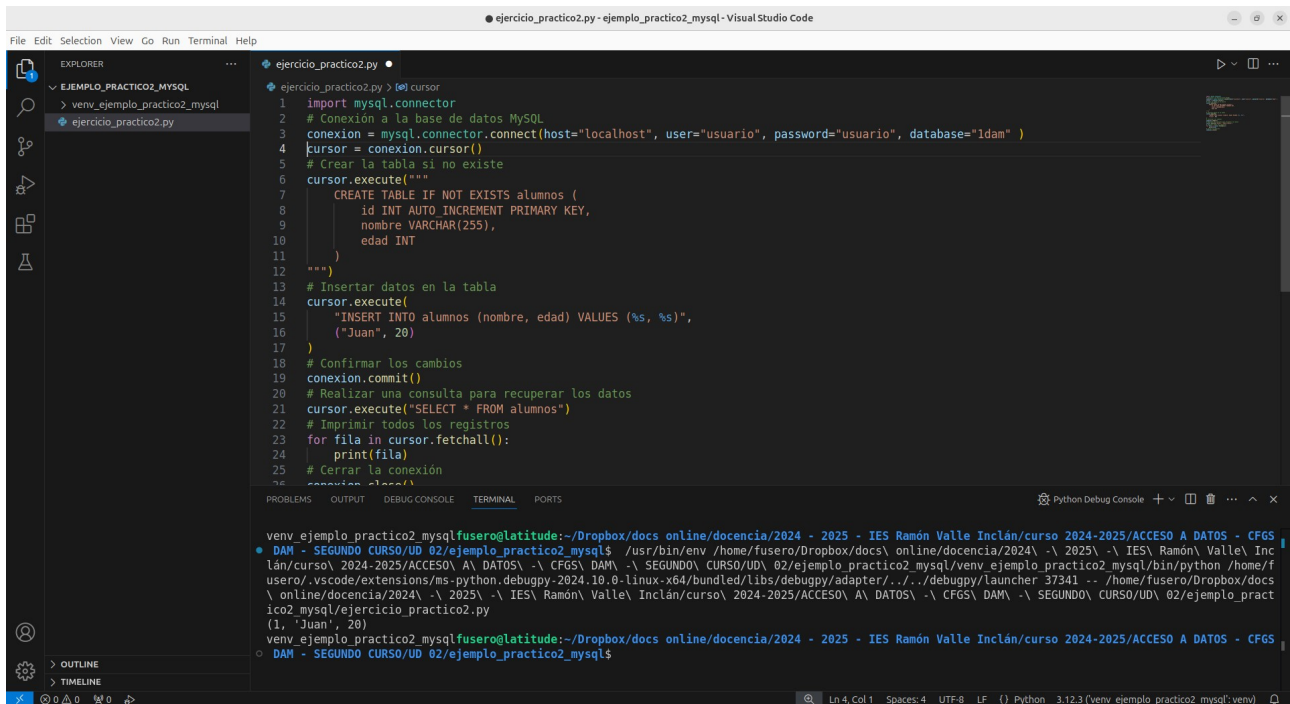
A continuación se muestra una captura de pantalla de cómo quedaría la ejecución.

1Si te da el error siguiente:

```
mysql.connector.errors.NotSupportedError: Authentication plugin 'caching_sha2_password' is not supported
```

Se arregla actualizando a la última versión del conector:

```
pip install --upgrade mysql-connector-python
```



```
ejercicio_practico2.py - ejemplo_practico2_mysql - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
  EJEMPLO_PRACTICO2_MYSQL
    venv_ejemplo_practico2_mysql
      ejercicio_practico2.py

ejercicio_practico2.py
1 import mysql.connector
2 # Conexión a la base de datos MySQL
3 conexion = mysql.connector.connect(host="localhost", user="usuario", password="usuario", database="ldam")
4 cursor = conexion.cursor()
5 # Crear la tabla si no existe
6 cursor.execute("""
7     CREATE TABLE IF NOT EXISTS alumnos (
8         id INT AUTO INCREMENT PRIMARY KEY,
9         nombre VARCHAR(255),
10        edad INT
11    )
12 """)
13 # Insertar datos en la tabla
14 cursor.execute(
15     "INSERT INTO alumnos (nombre, edad) VALUES (%s, %s)",
16     ("Juan", 20)
17 )
18 # Confirmar los cambios
19 conexion.commit()
20 # Realizar una consulta para recuperar los datos
21 cursor.execute("SELECT * FROM alumnos")
22 # Imprimir todos los registros
23 for fila in cursor.fetchall():
24     print(fila)
25 # Cerrar la conexión
26 conexion.close()

TERMINAL
venv_ejemplo_practico2_mysql fusero@latitudo:~/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CF6S
DAM - SEGUNDO CURSO/UD 02/ejemplo_practico2_mysql$ /usr/bin/env /home/fusero/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CF6S DAM - SEGUNDO CURSO/UD 02/ejemplo_practico2_mysql/venv_ejemplo_practico2_mysql/bin/python /home/fusero/.vscode/extensions/ms-python.debugpy-2024.10.0-linux.x64/bundled/libs/debugpy/adapter/../../debugpy/launcher 37341 -- /home/fusero/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CF6S DAM - SEGUNDO CURSO/UD 02/ejemplo_practico2_mysql/ejercicio_practico2.py
(1, 'Juan', 20)
venv_ejemplo_practico2_mysql fusero@latitudo:~/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CF6S
DAM - SEGUNDO CURSO/UD 02/ejemplo_practico2_mysql$
```

Explicación de los cambios

1. Conexión a MySQL:

- Utilizamos `mysql.connector.connect()` en lugar de `sqlite3.connect()`. Aquí necesitas proporcionar:

- `host`: El servidor donde está la base de datos (usualmente `localhost`).
- `user`: El usuario con el que te conectas a MySQL.
- `password`: La contraseña del usuario.
- `database`: El nombre de la base de datos que ya debe existir en MySQL.

2. Consultas SQL:

- El formato de las consultas SQL no cambia mucho. Solo debes asegurarte de que los tipos de datos sean compatibles con MySQL.

- Por ejemplo, `VARCHAR(255)` es utilizado en lugar de `TEXT` para almacenar cadenas.

3. Marcadores de posición:

- En MySQL, los marcadores de posición para las consultas `INSERT` y otras consultas parametrizadas usan `%s` en lugar de `?`, pero la función sigue siendo la misma.

4. Confirmación de los cambios:

- El uso de ``conexion.commit()`` es igual que en SQLite, lo que asegura que los cambios (inserciones, actualizaciones, eliminaciones) se guarden en la base de datos.

5. Recuperar y mostrar los datos:

- Utilizamos el cursor para ejecutar la consulta ``SELECT`` y luego recorremos los resultados con un bucle para imprimirlos.

Actividad 2 de clase.

Haz el mismo script de MySQL, pero usando el conector PyMySQL en lugar de mysql connector. El programa ha de acceder a la tabla de objetos que te haya tocado en el reparto.

Adjunta captura de pantalla de Visual Studio Code en el que se vea el código y la traza de ejecución del programa en el terminal incrustado de Code.

3. Selección del conector idóneo en la aplicación

En el apartado anterior hemos tratado dos scripts que usan conectores distintos, hacen lo mismo, son muy similares en cuanto a sintaxis, pero hay algunas diferencias técnicas clave entre los conectores **mysql-connector** y **pymysql**:

1. Desarrolladores:

- **mysql-connector** es un conector oficial desarrollado y mantenido por Oracle, la empresa que mantiene MySQL.
- **pymysql** es una implementación de terceros desarrollada por la comunidad Python y es una alternativa ligera para conectarse a bases de datos MySQL.

2. Compatibilidad:

- **mysql-connector** es parte del ecosistema oficial de MySQL, lo que significa que está totalmente alineado con las últimas actualizaciones de MySQL.
- **pymysql** es una alternativa compatible con MySQL, pero es independiente de Oracle. Sin embargo, también soporta la mayoría de las características esenciales de MySQL y es una opción popular en la comunidad Python.

3. Instalación:

- Para **mysql-connector**, usualmente se instala con `pip install mysql-connector-python`.
- Para **pymysql**, se instala con `pip install pymysql`.

¿Cuál escoger?

- Si prefieres usar el conector oficial de MySQL, **mysql-connector** puede ser una mejor opción. Tiene más soporte directo de Oracle y es fácil de usar.
- Si prefieres una opción ligera, más comúnmente usada en entornos con SQLAlchemy o necesitas una alternativa completamente desarrollada por la comunidad, entonces **pymysql** es una excelente opción.

Ambos ofrecen capacidades similares para proyectos educativos o de desarrollo en Python que interactúan con bases de datos MySQL.

Actividad 3 de clase.

Objetivos:

- Comparar el rendimiento de dos conectores (mysql-connector y pymysql) para MySQL en Python.
- Medir el tiempo de ejecución de operaciones CRUD en una base de datos usando ambos conectores.
- Analizar los resultados y discutir cómo el conector utilizado puede afectar el rendimiento en aplicaciones a gran escala.

Instrucciones:

Los alumnos deberán usar la base de datos “1dam”. Cada alumno usará la tabla que creó en actividades anteriores, correspondiente al tipo de objeto que se le ha asignado (películas, coches, etc.).

Los alumnos deberán escribir 4 scripts:

(A) Dos de ellos relizarán una **inserción masiva de datos** (10.000 veces) en la tabla. Se hará la misma operación usando **mysql-connector** y **pymysql**, midiendo el tiempo que toma cada operación. Para medir el tiempo podemos usar algo como esto:

```
import time
start_time = time.time()
#Hacemos operaciones
end_time = time.time()
print(f"Tiempo de inserción con mysql-connector: {end_time - start_time} segundos")
```

Los datos a insertar deben ser aleatorios. Un ejemplo del código a usar en el caso de Herramientas es:

```
tipos = ['Manual', 'Eléctrica', 'Neumática', 'Hidráulica']
materiales = ['Acero', 'Plástico', 'Aluminio', 'Hierro']
usos = ['Corte', 'Perforación', 'Sujeción', 'Medición']
marcas = ['MarcaA', 'MarcaB', 'MarcaC', 'MarcaD']
.....
nombre = f"Herramienta {i+1}"
tipo = random.choice(tipos)
material = random.choice(materiales)
uso = random.choice(usos)
```

(B) Los otros dos scripts realizarán una lectura masiva (ejecutar select * 10.000 veces), con los dos conectores.

Cada script se ejecutará un mínimo de dos veces.

Al final de esta entrega el alumnado debe:

(1) Mostrar unos datos de benchmark similares a estos:

Tiempo de inserción con mysql-connector: 1.3059463500976562 segundos

Tiempo de inserción con pymysql: 1.2221815586090088 segundos

Tiempo de inserción con pymysql: 1.0650999546051025 segundos

Tiempo de inserción con pymysql: 1.0478477478027344 segundos

Tiempo de consulta con mysql-connector: 0.07146072387695312 segundos

Tiempo de consulta con mysql-connector: 0.07426619529724121 segundos

Tiempo de consulta con pymysql: 0.39804625511169434 segundos

Tiempo de consulta con pymysql: 0.22675466537475586 segundos

(2) Adjuntar los ficheros relevantes de código python para evidenciar la creación de los 4 scripts.

Si no se entrega una de las dos cosas, la actividad no será tomada en cuenta a efectos de evaluación ni de calificación.

4. Definición de la estructura de la base de datos

Objetivo:

Aprender a diseñar y crear la estructura de una base de datos relacional, incluyendo tablas y relaciones entre ellas.

Explicación Teórica:

Una base de datos relacional está compuesta por tablas, donde cada tabla tiene columnas (campos) y filas (registros). Es importante definir adecuadamente el tipo de dato de cada campo, las relaciones entre tablas (usando claves foráneas), y las restricciones (como NOT NULL, UNIQUE, etc.).

En MySQL, la estructura se define a través de sentencias SQL como CREATE DATABASE y CREATE TABLE, mientras que en Python, usamos conectores como mysql-connector-python para ejecutar esas sentencias directamente desde nuestros programas.

A continuación, exploramos cómo definir la estructura de la base de datos desde Python, con ejemplos prácticos para guiarte en el proceso, obviando la parte de la creación de la base de datos “1dam” y de la tabla “Herramientas”.

A continuación, un ejemplo de cómo definir una relación entre una tabla **Herramientas** y una tabla **Proveedores**, donde cada herramienta puede tener un proveedor asociado:

```
# Crear la tabla 'Proveedores'
sql_crear_proveedores = """
CREATE TABLE IF NOT EXISTS Proveedores (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    direccion VARCHAR(100)
)
"""

cursor.execute(sql_crear_proveedores)

# Añadir la columna 'proveedor_id' a la tabla Herramientas
sql_alter_herramientas = """
ALTER TABLE Herramientas ADD proveedor_id INT,
ADD CONSTRAINT fk_proveedor
FOREIGN KEY (proveedor_id) REFERENCES Proveedores(id)
"""

cursor.execute(sql_alter_herramientas)

print("Relación entre Herramientas y Proveedores creada.")
```

En este ejemplo, cada herramienta puede tener un proveedor asociado a través de la columna `proveedor_id`, que es una clave foránea que hace referencia al campo `id` de la tabla **Proveedores**.

Actividad 4 de clase

Cada `alumn@` va a diseñar una tabla adicional a la tabla que ya tiene asignada. Se deja a elección del alumnado la elección del objeto que represente la tabla adicional. Las dos tablas tendrán una relación 1:n. Lo va a hacer con el conector *pymysql*, ejecutando un mínimo de dos instrucciones SQL nuevas: 1 para la creación de la tabla y otra para la definición de la restricción (clave foránea). Adjuntará tantas capturas de pantalla como sea necesario para ver el código y la traza de la ejecución.

5. Modificación del contenido de la base de datos

Objetivo:

Aprender a modificar registros existentes en la base de datos utilizando Python.

Explicación Teórica:

Las operaciones de modificación en una base de datos incluyen actualizaciones (`UPDATE`) y eliminaciones (`DELETE`). Es importante manejar estas operaciones con cuidado para no alterar datos erróneamente.

Actividad 5 de clase:

1. Modificar registros existentes:

El alumnado debe escribir un programa que modifique los registros de la tabla que se le haya asignado a cada uno, actualizando un campo en concreto de una sólo fila.

Código Ejemplo (Actualizar datos):

```
cursor.execute("UPDATE alumnos SET edad = %s WHERE nombre = %s", (25, 'Juan'))
conexion.commit()
print(cursor.rowcount, "registro(s) actualizado(s)")
```

2. Eliminar registros: El alumnado debe escribir un programa que elimine un(os) registro(s) de la base de datos asignada a cada persona, basado en condiciones específicas (por ejemplo, eliminar todos los alumnos mayores de 30 años).

Código Ejemplo (Eliminar datos):

```
cursor.execute("DELETE FROM alumnos WHERE edad > %s", (30,))
conexion.commit()
print(cursor.rowcount, "registro(s) eliminado(s)")
```

Adjunta:

→ Dos capturas de pantalla de Visual Code, una para la actualización y otra para el borrado. Cada captura debe mostrar el código y el resultado de la ejecución

→ En el caso de la actualización, muestra una captura de pantalla de la consola de MySQL que muestre el antes y el después.

→ En el caso del borrado, una captura adicional (también de la consola MySQL) que muestre todos los registros de la tabla antes y después de borrar un registro.

4 capturas de pantalla en total. **Si no se entrega una de las 4 capturas, la actividad no será tenida en cuenta a efectos de evaluación ni de calificación.**

6. Definición de objetos para almacenar resultados de consultas

Al realizar consultas SQL (`SELECT`), los resultados se pueden almacenar en variables o estructuras de datos en Python, como listas o diccionarios, dependiendo del tipo de resultado esperado.

Métodos comunes de manejo de resultados:

- `fetchone()`: Recupera el siguiente registro del conjunto de resultados.
- `fetchall()`: Recupera todos los registros del conjunto de resultados. Continúa desde la posición actual del cursor, no desde el principio de la consulta. Esto significa que si previamente has usado `fetchone()` o `fetchmany()`, el cursor se mueve al leer las filas, y cuando llamas a `fetchall()`, este devolverá todas las filas restantes a partir de la posición actual del cursor.
- `fetchmany(size)`: Recupera un número específico de registros.

Tanto `fetchall()` como `fetchmany()` continúan desde la posición actual del cursor, no desde el principio de la consulta. Esto significa que si previamente has usado `fetchone()` o `fetchmany()`, el cursor se mueve al leer las filas, y cuando llamas a `fetchall()` o `fetchmany()`, este devolverá todas las filas **restantes** a partir de la posición actual del cursor.

Ejercicio Práctico 3: Uso de los métodos `fetchone()`, `fetchall()` y `fetchmany()`

Veamos un ejemplo que realizar distintas consultas que recuperes todas las herramienta de la tabla “Herramientas” de la base de datos `1dam` y los saque por pantalla. Este ejemplo usará los 3 métodos mencionados más arriba:

```
import mysql.connector

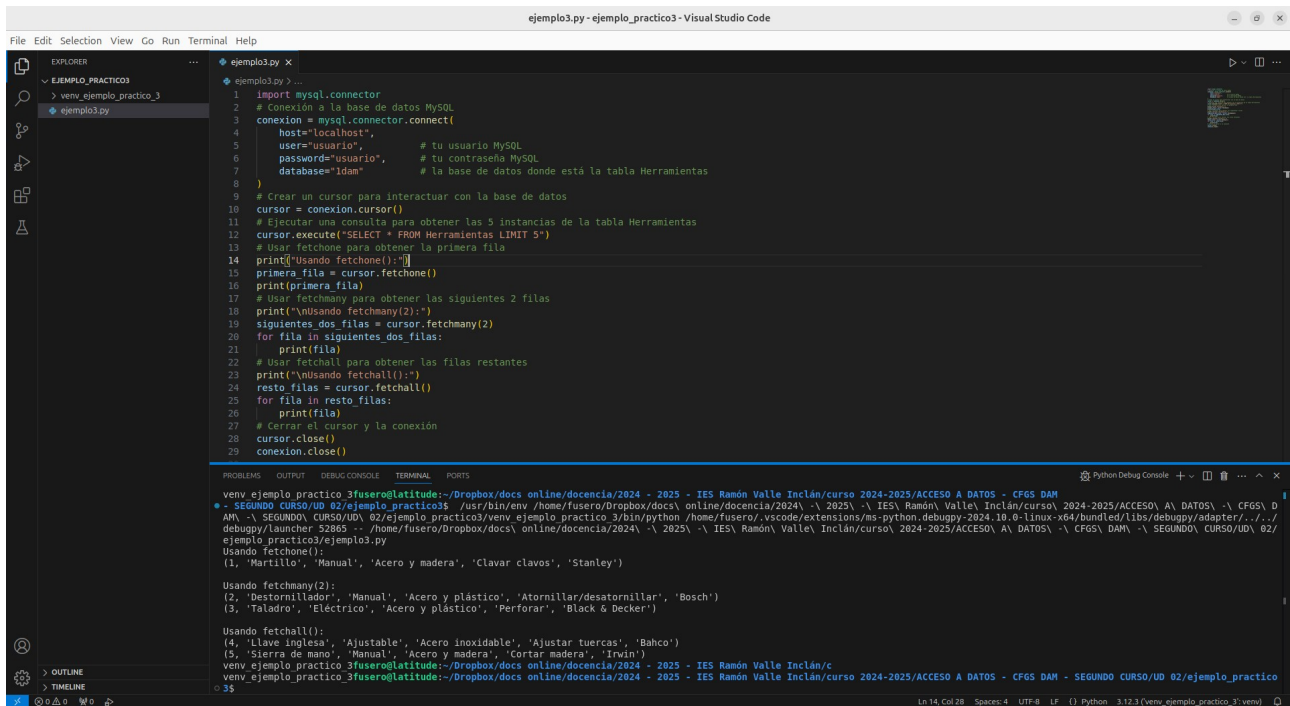
# Conexión a la base de datos MySQL
conexion = mysql.connector.connect(
    host="localhost",
    user="usuario",          # tu usuario MySQL
    password="usuario",      # tu contraseña MySQL
    database="1dam"          # la base de datos donde está la tabla Herramientas
```

```
)  
# Crear un cursor para interactuar con la base de datos  
cursor = conexion.cursor()  
# Ejecutar una consulta para obtener las 5 instancias de la tabla Herramientas  
cursor.execute("SELECT * FROM Herramientas LIMIT 5")  
# Usar fetchone para obtener la primera fila  
print("Usando fetchone():")  
primera_fila = cursor.fetchone()  
print(primera_fila)  
# Usar fetchmany para obtener las siguientes 2 filas  
print("\nUsando fetchmany(2):")  
siguientes_dos_filas = cursor.fetchmany(2)  
for fila in siguientes_dos_filas:  
    print(fila)  
# Usar fetchall para obtener las filas restantes  
print("\nUsando fetchall():")  
resto_filas = cursor.fetchall()  
for fila in resto_filas:  
    print(fila)  
# Cerrar el cursor y la conexión  
cursor.close()  
conexion.close()
```

[NOTA: Es crucial cerrar las conexiones a la base de datos y eliminar los objetos (como cursores) que ya no son necesarios. Esto libera recursos del sistema y previene posibles bloqueos o fugas de memoria. En muchas ocasiones estas operaciones se hacen dentro de la cláusula finally cuando estamos gestionando excepciones]

Vemos la traza de la ejecución:

Módulo profesional: Acceso a datos



```
ejemplo3.py - ejemplo_practico3 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER
  EJEMPLO_PRACTICO3
    venv_ejemplo_practico_3
      ejemplo3.py

ejemplo3.py X
1 import mysql.connector
2 # Conexión a la base de datos MySQL
3 conexion = mysql.connector.connect(
4     host="localhost",
5     user="usuario",      # tu usuario MySQL
6     password="usuario",  # tu contraseña MySQL
7     database="Idam"      # la base de datos donde está la tabla Herramientas
8 )
9 # Crear un cursor para interactuar con la base de datos
10 cursor = conexion.cursor()
11 # Ejecutar una consulta para obtener las 5 instancias de la tabla Herramientas
12 cursor.execute("SELECT * FROM Herramientas LIMIT 5")
13 # Usar fetchone para obtener la primera fila
14 print(f"Usando fetchone():")
15 primera_fila = cursor.fetchone()
16 print(primera_fila)
17 # Usar fetchmany para obtener las siguientes 2 filas
18 print(f"Usando fetchmany(2):")
19 siguientes_dos_filas = cursor.fetchmany(2)
20 for fila in siguientes_dos_filas:
21     print(fila)
22 # Usar fetchall para obtener las filas restantes
23 print(f"Usando fetchall():")
24 resto_filas = cursor.fetchall()
25 for fila in resto_filas:
26     print(fila)
27 # Cerrar el cursor y la conexión
28 cursor.close()
29 conexion.close()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
venv_ejemplo_practico_3fusero@latitude:~/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CFGS DAM
- SEGUNDO CURSO/UD 02/ejemplo_practico3 /usr/bin/env /home/fusero/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CFGS DAM - SEGUNDO CURSO/UD 02/ejemplo_practico3/bin/python /home/fusero/.vscode/extensions/ms-python/debugpy-2024.10.0-linux-x64/bundled/libs/debugpy/adapter/. / /
debugpy/launcher 52865 -- -- /home/fusero/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CFGS DAM - SEGUNDO CURSO/UD 02/
ejemplo_practico3/ejemplo3.py
Usando fetchone():
(1, 'Martillo', 'Manual', 'Acero y madera', 'Clavar clavos', 'Stanley')
Usando fetchmany(2):
(2, 'Destornillador', 'Manual', 'Acero y plástico', 'Atornillar/desatornillar', 'Bosch')
(3, 'Taladro', 'Eléctrico', 'Acero y plástico', 'Perforar', 'Black & Decker')
Usando fetchall():
(4, 'Llave inglesa', 'Ajustable', 'Acero inoxidable', 'Ajustar tuercas', 'Bahco')
(5, 'Sierra de mano', 'Manual', 'Acero y madera', 'Cortar madera', 'Irwin')
venv_ejemplo_practico_3fusero@latitude:~/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/c
venv_ejemplo_practico_3fusero@latitude:~/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CFGS DAM - SEGUNDO CURSO/UD 02/ejemplo_practico3$
```

Actividad 6 de clase:

Cada alumn@ partirá de la tabla que se le haya asignado (restaurantes, películas, etc.) con sólo 5 filas. Ahora cambiamos el conector a pymysql. Mostraremos los resultados uno a uno y cuando se terminen de mostrar los 5 resultados, mostraremos de nuevo dichos 5 resultados.

El programa sólo debe usar fetchone() y además no debe almacenar en ninguna estructura de datos los datos concernientes a más de una fila.

Adjunta tantas capturas de pantalla como sean necesarias para ver todo el código y la traza de la ejecución.

7. Gestión de transacciones

Objetivo:

Comprender y aplicar el concepto de transacciones, asegurando la integridad de los datos.

Explicación Teórica:

Una transacción es un conjunto de operaciones SQL que se ejecutan de manera atómica, es decir, o se completan todas o ninguna. Las transacciones permiten asegurar la consistencia de los datos en caso de errores.

- COMMIT: Confirma todas las operaciones realizadas en la transacción.
- ROLLBACK: Revierte las operaciones realizadas si ocurre un error.

Ejercicio Práctico 4: Implementar una transacción:

Programa Python que utiliza **transacciones** con el conector mysql-connector para añadir un registro a la tabla **Herramientas**. El programa gestionará las **excepciones** de manera que, si la operación se realiza con éxito, se ejecutará un **commit**, y si ocurre algún error, se realizará un **rollback** para deshacer los cambios. Además, se cerrarán las conexiones y destruirán los objetos que no se vayan a usar más.

```
import mysql.connector
from mysql.connector import Error
# Conexión a la base de datos MySQL usando el conector oficial
conexion = None
try:
    # Conectar a la base de datos
    conexion = mysql.connector.connect(
        host="localhost",
        user="usuario",          # tu usuario MySQL
        password="usuario",      # tu contraseña MySQL
        database="ldam"          # la base de datos donde está la tabla Herramientas
    )
    if conexion.is_connected():
        # Crear un cursor
        cursor = conexion.cursor()
        # Iniciar la transacción
        print("Iniciando transacción...")
        # Insertar un nuevo registro en la tabla Herramientas
```

```
sql_insert = """
    INSERT INTO Herramientas (nombre, tipo, peso, marca)
    VALUES (%s, %s, %s, %s)
    """

datos_herramienta = ("Cinzel", "Manual", "0.5kg", "Stanley")
cursor.execute(sql_insert, datos_herramienta)

# Hacer commit si todo va bien
conexion.commit()

print("Transacción exitosa: Registro insertado correctamente.")

except Error as e:

    # Si ocurre un error, hacer rollback
    print(f"Error en la transacción: {e}")

    if conexion:
        conexion.rollback()
        print("Se realizó rollback.")


finally:

    # Cerrar el cursor y la conexión si están abiertos
    if conexion and conexion.is_connected():
        cursor.close()
        conexion.close()
        print("Conexión cerrada.")
```

Actividad 7 de clase:

El alumnado usará un script en Python similar al anterior, pero forzará un fallo, de forma que se haga el rollback. Se deja a elección del alumnado el fallo que se cometerá a propósito.

Adjuntar captura de pantalla de Visual Studio Code en la que se vea tanto el programa como la traza de la ejecución:



```
Iniciando transacción...
Error en la transacción:
Se realizó rollback.
Conexión cerrada.
```


8. Ejecución de procedimientos almacenados

Objetivo:

Ejecutar procedimientos almacenados desde una aplicación Python.

Explicación Teórica:

Los procedimientos almacenados son conjuntos de instrucciones SQL predefinidos que se almacenan en la base de datos y pueden ser ejecutados por la aplicación. Son útiles para operaciones repetitivas o complejas, y mejoran el rendimiento al reducir la cantidad de tráfico entre la aplicación y la base de datos.

Ejercicio Práctico 5: Crear y ejecutar un procedimiento almacenado:

Creamos un procedimiento almacenado en MySQL que tenga una consulta SELECT COUNT con una condición.

Código SQL para crear el procedimiento:

```
DELIMITER //
```

```
CREATE PROCEDURE contar_herramientas()
```

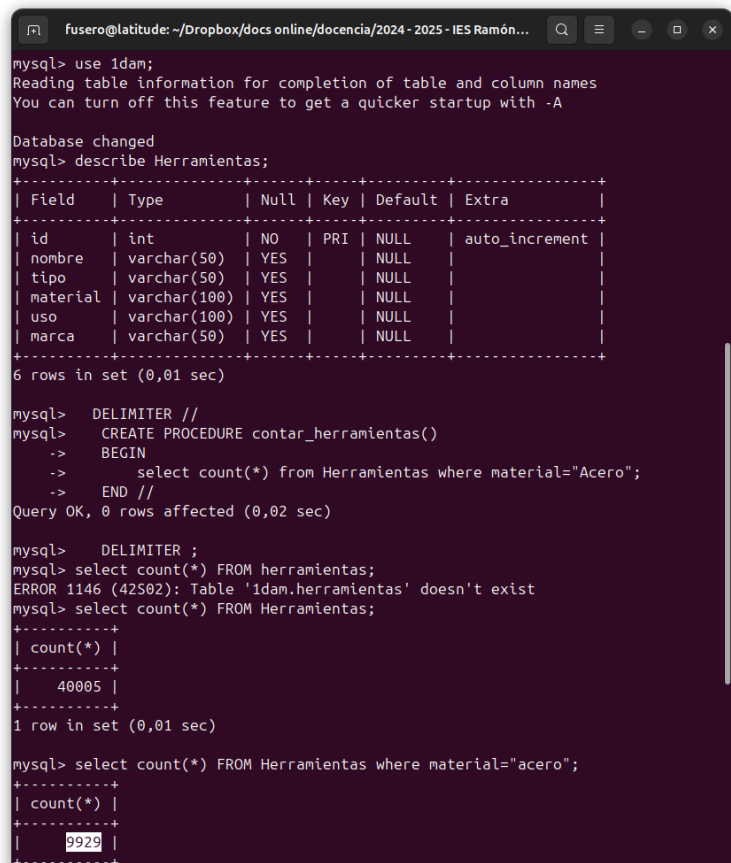
```
BEGIN
```

```
    select count(*) from Herramientas where material="Acero";
```

```
END //
```

```
DELIMITER ;
```

Se puede ver en esta captura de pantalla como se crea el procedimiento en la consola de MYSQL directamente y como se verifica que el resultado de la transacción debe dar 9929.



```
fusero@latitude: ~/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón...
mysql> use 1dam;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> describe Herramientas;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id     | int  | NO   | PRI | NULL    | auto_increment |
| nombre | varchar(50) | YES | | NULL    | |
| tipo   | varchar(50) | YES | | NULL    | |
| material | varchar(100) | YES | | NULL    | |
| uso    | varchar(100) | YES | | NULL    | |
| marca  | varchar(50) | YES | | NULL    | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0,01 sec)

mysql> DELIMITER //
mysql> CREATE PROCEDURE contar_herramientas()
-> BEGIN
->     select count(*) from Herramientas where material="Acero";
-> END //
Query OK, 0 rows affected (0,02 sec)

mysql> DELIMITER ;
mysql> select count(*) FROM herramientas;
ERROR 1146 (42S02): Table '1dam.herramientas' doesn't exist
mysql> select count(*) FROM Herramientas;
+-----+
| count(*) |
+-----+
|      40005 |
+-----+
1 row in set (0,01 sec)

mysql> select count(*) FROM Herramientas where material="acero";
+-----+
| count(*) |
+-----+
|      9929 |
+-----+
```

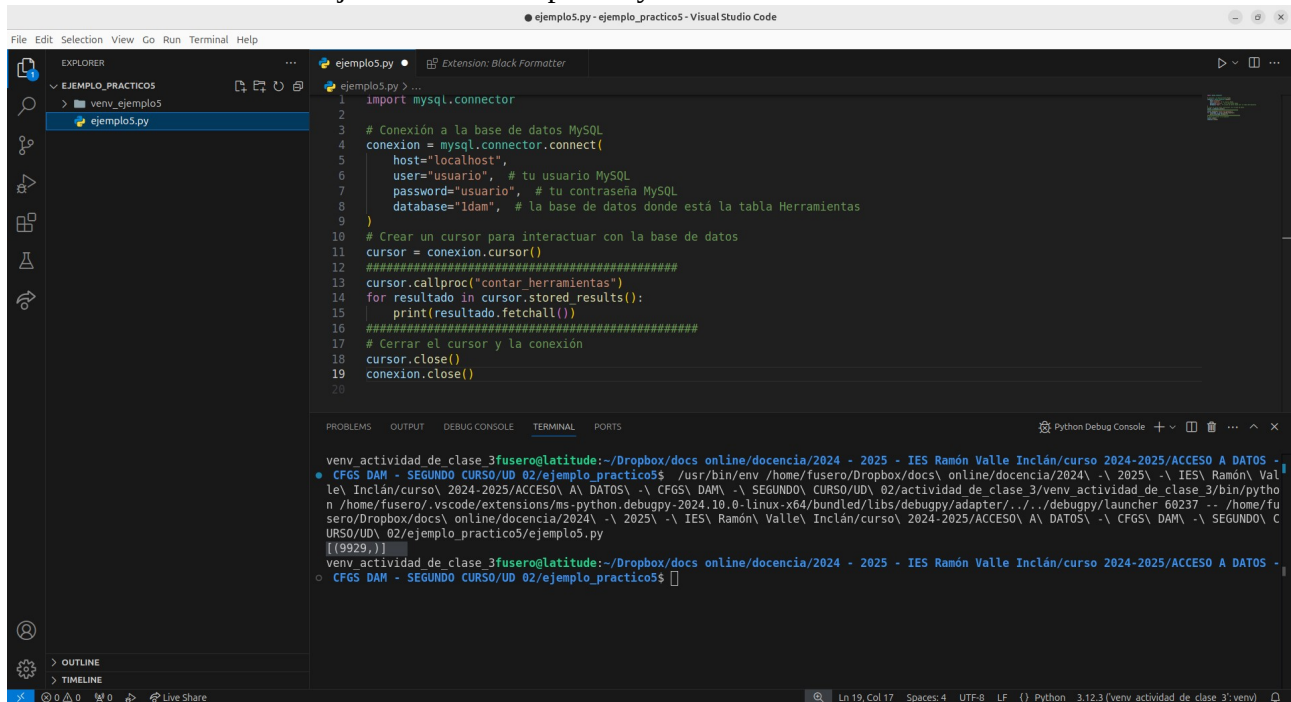
Código python para ejecutar el procedimiento:

```
cursor.callproc('contar_herramientas')

for resultado in cursor.stored_results():

    print(resultado.fetchall())
```

Y así se ve la traza de la ejecución del script de Python:



The screenshot shows the Visual Studio Code interface. The Explorer pane on the left shows a file named `ejemplo5.py` under the `venv_ejemplo5` folder. The main editor displays the content of `ejemplo5.py`, which is a Python script that connects to a MySQL database, executes a stored procedure, and prints the results. The script includes comments in Spanish explaining each step. The Output pane at the bottom shows the execution of the script, with the output of the `print` statement visible.

```
1 import mysql.connector
2
3 # Conexión a la base de datos MySQL
4 conexion = mysql.connector.connect(
5     host="localhost",
6     user="usuario", # tu usuario MySQL
7     password="usuario", # tu contraseña MySQL
8     database="ldam", # la base de datos donde está la tabla Herramientas
9 )
10 # Crear un cursor para interactuar con la base de datos
11 cursor = conexion.cursor()
12 #####
13 cursor.callproc("contar_herramientas")
14 for resultado in cursor.stored_results():
15     print(resultado.fetchall())
16 #####
17 # Cerrar el cursor y la conexión
18 cursor.close()
19 conexion.close()
20
```

venv_actividad_de_clase_3fusero@latitude:~/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CF6S DAM - SEGUNDO CURSO/UD 02/ejemplo_practico5\$ /usr/bin/env /home/fusero/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CF6S DAM - SEGUNDO CURSO/UD 02/ejemplo_practico5/bin/python /home/fusero/.vscode/extensions/ms-python.debugpy-2024.10.0-linux-x64/bundled/libs/debugpy/adapter/../../debugpy/launcher 60237 -- /home/fusero/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CF6S DAM - SEGUNDO CURSO/UD 02/ejemplo_practico5/ejemplo5.py

[[9929,]]

venv_actividad_de_clase_3fusero@latitude:~/Dropbox/docs online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CF6S DAM - SEGUNDO CURSO/UD 02/ejemplo_practico5\$

El método `stored_results()`

El método `stored_results()` se utiliza en el conector oficial de MySQL para obtener los resultados de un procedimiento almacenado después de que ha sido ejecutado con `callproc()`.

Cómo funciona `stored_results()`:

1. Cuando ejecutas un procedimiento almacenado en MySQL usando el método `callproc()`, no se devuelve inmediatamente un resultado como cuando ejecutas una consulta SQL con `execute()`. En cambio, `callproc()` devuelve un generador con los resultados que debes procesar manualmente.
2. Para acceder a los resultados de un procedimiento almacenado, usas el método `stored_results()`, que devuelve un generador. Este generador permite acceder a los diferentes conjuntos de resultados que el procedimiento pueda haber producido.
3. Usualmente, cada conjunto de resultados de un procedimiento almacenado es tratado como un cursor, y puedes iterar a través de los resultados utilizando el generador de `stored_results()`.

Pasos clave en el uso de `stored_results()`:

1. `callproc()`: Se usa para llamar a un procedimiento almacenado en MySQL.
2. `stored_results()`: Después de ejecutar el procedimiento almacenado con `callproc()`, usas este método para obtener los resultados devueltos por el procedimiento. Este método devuelve un generador, el cual puedes recorrer.
3. `fetchall()`, `fetchone()`, `fetchmany()`: Puedes usar estos métodos (como con cualquier consulta SQL) para recuperar los resultados devueltos.

¿Cuándo usar `stored_results()`?

- Procedimientos almacenados: Cuando estás trabajando con procedimientos almacenados en MySQL y necesitas obtener los resultados que estos generan, ya que `callproc()` no devuelve los resultados directamente.
- Resultados múltiples: En casos donde un procedimiento almacenado genera múltiples conjuntos de resultados (por ejemplo, cuando ejecuta varias consultas `SELECT` en su interior), puedes iterar sobre ellos con `stored_results()`.

Ejemplo de múltiples conjuntos de resultados:

Si tienes un procedimiento que devuelve múltiples resultados, por ejemplo:

```
DELIMITER //
```

```
CREATE PROCEDURE obtener_datos()  
BEGIN  
    SELECT COUNT(*) AS total_herramientas FROM Herramientas;  
    SELECT nombre FROM Herramientas LIMIT 5;  
END //
```

```
DELIMITER ;
```

Aquí, el procedimiento almacenado devuelve dos conjuntos de resultados: el primero es un conteo de herramientas y el segundo es una lista de nombres de las primeras 5 herramientas.

Para obtener ambos conjuntos de resultados en Python, puedes hacer lo siguiente:

```
....  
# Llamar al procedimiento almacenado  
cursor.callproc("obtener_datos")  
  
# Obtener los múltiples conjuntos de resultados  
for resultado in cursor.stored_results():  
    datos = resultado.fetchall() # Recuperar todas las filas del conjunto actual  
    print(datos)  
.....
```

Actividad 8 de clase.

El alumnado creará un procedimiento almacenado que realice una operación de complejidad similar. Lo ejecutará desde Python. Adjunta 2 capturas de pantalla:

- 1) Code con el código y la traza
- 2) Consola de Mysql con el procedimiento creado.

El ejercicio puntuará doble si el procedimiento almacenado acepta algún parámetro.

Anexo A. Virtualenv

¿Qué es `virtualenv`?

`virtualenv` es una herramienta que permite crear entornos virtuales en Python. Un entorno virtual es un directorio que contiene una instalación aislada de Python, junto con un conjunto independiente de paquetes y librerías. Esto es muy útil porque permite que cada proyecto tenga sus propias dependencias, sin interferir con otros proyectos en el mismo sistema.

¿Para qué sirve `virtualenv`?

La principal utilidad de `virtualenv` es la **gestión de dependencias**. Cuando trabajas en varios proyectos a la vez, puede que cada uno necesite diferentes versiones de los mismos paquetes. Si no utilizas un entorno virtual, las dependencias de un proyecto podrían causar problemas en otro. Usando `virtualenv`, cada proyecto tiene su propio "mini Python" con su propio conjunto de paquetes, lo que evita conflictos entre proyectos.

Ventajas de `virtualenv`:

- Aislamiento de dependencias: Cada entorno virtual contiene sus propias librerías y paquetes.
- Múltiples versiones: Puedes tener diferentes versiones de la misma librería en distintos proyectos.
- Fácil de usar: Proporciona comandos sencillos para crear y activar entornos virtuales.
- Mantenimiento de proyectos: Facilita compartir entornos de trabajo consistentes con otros programadores.

Ejemplo Práctico: Uso de virtualenv

1. Instalar virtualenv:

```
apt install virtualenv
```

2. Crear un entorno virtual:

Supongamos que estamos creando un proyecto de gestión de alumnos. Para crear un entorno virtual en la carpeta del proyecto, ejecutamos:

```
mkdir alumnos
```

```
cd alumnos
```

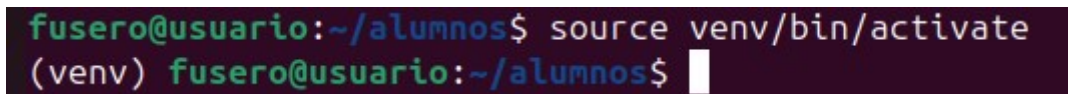
```
virtualenv venv
```

Esto creará una carpeta llamada `venv` dentro de `alumnos`, que contendrá una instalación aislada de Python y sus paquetes.

3. Activar el entorno virtual:

```
source venv/bin/activate
```

Al activarse, verás que el nombre del entorno virtual aparece al principio del terminal entre paréntesis, lo que indica que estás dentro del entorno.



```
fusero@usuario:~/alumnos$ source venv/bin/activate
(venv) fusero@usuario:~/alumnos$
```

4. Instalar dependencias:

Ahora puedes instalar las dependencias específicas de tu proyecto sin preocuparte por afectar otros proyectos:

```
pip install mysql-connector-python
```

5. Desactivar el entorno virtual:

Cuando termines de trabajar, puedes salir del entorno virtual con:

```
deactivate
```

¿Qué ocurre si no usamos `virtualenv`?

Si no utilizas `virtualenv`, todas las dependencias de tus proyectos se instalarán globalmente en tu sistema. Esto genera varios problemas:

- Conflicto de versiones: Si un proyecto necesita la versión 1.2 de una librería y otro proyecto necesita la versión 2.0, no podrías instalarlas ambas al mismo tiempo sin `virtualenv`. El último proyecto que instale la librería sobrescribiría la versión anterior.
- Inconsistencia entre desarrolladores: Si compartes el proyecto con otros programadores, no tendrías un método estandarizado para compartir las versiones exactas de las dependencias que estás utilizando. Cada programador podría tener una configuración diferente, lo que provocaría errores difíciles de rastrear.

Aquí tienes un ejemplo más concreto que muestra cómo `virtualenv` permite trabajar con diferentes versiones de una misma librería en distintos proyectos:

Supongamos que tienes dos proyectos diferentes, ambos conectados a bases de datos MySQL, pero uno necesita la versión 8.0.29 de la librería `mysql-connector-python`, mientras que el otro requiere la versión 8.0.31. Sin `virtualenv`, tendrías que instalar una versión globalmente, lo que podría causar conflictos entre ambos proyectos. Vamos a ver cómo `virtualenv` soluciona este problema.

1. Creamos un entorno virtual para cada proyecto.

- Para el proyecto 1, que necesita la versión 8.0.29 de `mysql-connector-python`:

```
mkdir proyecto1
cd proyecto1
virtualenv venv_proyecto1
```

- Para el proyecto 2, que necesita la versión 8.0.31 de `mysql-connector-python`:

```
mkdir proyecto2
cd proyecto2
virtualenv venv_proyecto2
```

3. Activamos el entorno virtual y instalamos las versiones específicas:

- En el proyecto 1:

```
cd proyecto1
source venv_proyecto1/bin/activate
pip install mysql-connector-python==8.0.29
```

Verificamos la instalación:

```
pip freeze
# mysql-connector-python==8.0.29
```

- En el proyecto 2:

```
cd ../proyecto2
source venv_proyecto2/bin/activate
```

```
pip install mysql-connector-python==8.0.31
```

Verificamos la instalación:

```
pip freeze  
# mysql-connector-python==8.0.31
```

4. Comprobación del aislamiento de entornos:

- Si revisas las versiones instaladas en cada entorno usando `pip freeze`, verás que cada entorno virtual tiene su propia versión de la librería, sin interferir con los demás proyectos.

En el entorno virtual de proyecto 1:

```
pip freeze  
# mysql-connector-python==8.0.29
```

En el entorno virtual de proyecto 2:

```
pip freeze  
# mysql-connector-python==8.0.31
```

Cuando trabajas en el proyecto 1, estarás usando la versión 8.0.29, y cuando cambies al proyecto 2 usando su entorno virtual, estarás usando la versión 8.0.31. Esto es posible gracias al aislamiento de entornos que proporciona `virtualenv`.

5. Desactivar el entorno virtual:

Para salir del entorno virtual en cualquier momento, solo necesitas ejecutar:

```
deactivate
```

¿Qué ocurriría sin `virtualenv`?

Si no estuviéramos usando `virtualenv`, tendríamos que instalar las dependencias globalmente en el sistema. Veamos cómo se complicaría la situación:

1. Instalarías `mysql-connector-python` para proyecto 1:

```
pip install mysql-connector-python==8.0.29
```

2. Luego, cuando comiences a trabajar en proyecto 2, necesitarías actualizar la versión de la librería:

```
pip install mysql-connector-python==8.0.31
```


3. Ahora, cuando vuelvas a trabajar en proyecto 1, este proyecto podría romperse porque esperarías la versión 8.0.29, pero tienes instalada la versión 8.0.31. Esto generaría un conflicto de versiones que tendrías que gestionar manualmente cada vez que cambias de proyecto, reinstalando versiones constantemente.

Este proceso es tedioso y propenso a errores, ya que podrías olvidar reinstalar la versión correcta, rompiendo la aplicación o generando errores inesperados.

Este ejemplo te muestra de manera concreta cómo `virtualenv` te permite trabajar con diferentes versiones de una misma librería en proyectos distintos sin complicaciones.

Conclusión:

`virtualenv` simplifica la gestión de entornos y dependencias, permitiendo a los programadores centrarse en el desarrollo sin preocuparse por los conflictos entre librerías. En el contexto del desarrollo de aplicaciones con acceso a bases de datos en Python, `virtualenv` es una herramienta fundamental para asegurar que los proyectos se mantengan independientes y bien organizados.

Anexo B. Puesta en marcha del servidor de MySQL en Ubuntu 24.04

Seguir lo indicado en esta guía:

<https://voidnull.es/instalar-mysql-8-0-en-ubuntu-24-04/>

El último de la guía es ejecutar el script de securización, cuya tarza dejo aquí como guía para ver lo que hay que responder:

```
# mysql_secure_installation
```

```
Securing the MySQL server deployment.
```

```
Connecting to MySQL using a blank password.
```

```
VALIDATE PASSWORD COMPONENT can be used to test passwords
```

```
and improve security. It checks the strength of password
```

```
and allows the users to set only those passwords which are
```

```
secure enough. Would you like to setup VALIDATE PASSWORD component?
```

```
Press y|Y for Yes, any other key for No: y
```

```
There are three levels of password validation policy:
```

```
LOW      Length >= 8
```

```
MEDIUM Length >= 8, numeric, mixed case, and special characters
```

```
STRONG Length >= 8, numeric, mixed case, special characters and dictionary file
```

```
Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG: 0
```

```
Skipping password set for root as authentication with auth_socket is used by default.
```

```
If you would like to use password authentication instead, this can be done with the "ALTER_USER" command.
```

```
See https://dev.mysql.com/doc/refman/8.0/en/alter-user.html#alter-user-password-management for more information.
```

```
By default, a MySQL installation has an anonymous user,
allowing anyone to log into MySQL without having to have
a user account created for them. This is intended only for
testing, and to make the installation go a bit smoother.
You should remove them before moving into a production
```

environment.

Remove anonymous users? (Press y|Y for Yes, any other key for No) : n

... skipping.

Normally, root should only be allowed to connect from 'localhost'. This ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? (Press y|Y for Yes, any other key for No) : n

... skipping.

By default, MySQL comes with a database named 'test' that anyone can access. This is also intended only for testing, and should be removed before moving into a production environment.

Remove test database and access to it? (Press y|Y for Yes, any other key for No) : n

... skipping.

Reloading the privilege tables will ensure that all changes made so far will take effect immediately.

Reload privilege tables now? (Press y|Y for Yes, any other key for No) : y

Success.

B.1. Cambio de la clave de root

Después de esto cambiamos la clave de root del servidor de MYSQL, le vamos a poner la clave “usuario”.

1. Abrir el terminal o la consola de comandos de tu sistema.
2. Iniciar sesión en MySQL con el usuario `root`. Si ya tienes la contraseña anterior, puedes hacerlo con el siguiente comando:

```
mysql -u root -p
```

(si esto no te funciona debes intentarlo con sudo delante)

3. Introducir la contraseña actual cuando se te solicite.

4. Una vez que estés dentro de la consola de MySQL, ejecuta el siguiente comando para cambiar la contraseña de `root`:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'nueva_contraseña';
```

- Reemplaza `nueva_contraseña` por la nueva contraseña que desees establecer.(**usuario**).

5. Actualizar los privilegios para que los cambios tengan efecto:

```
FLUSH PRIVILEGES;
```

6. Salir de MySQL:

```
EXIT;
```

B.2. Creación de una base de datos y una tabla

Cada alumno tendrá asignada un objeto con el que va a trabar a lo largo del curso: Libros, Películas, Coches, Móviles, Restaurantes, Juguetes, Ropa, Equipos deportivos, Músicos, Viajes, Mascotas, Videojuegos, Obras de arte, Edificios históricos, Instrumentos musicales, Plantas, Vehículos eléctricos, Comidas, Series de TV, Bicicletas, Aplicaciones móviles, Herramientas. El reparto de tablas entre los alumnos se mostrará en clase.

El profesor tiene asignada la tabla herramientas. La base de datos se denominará “1dam” para todo el mundo.

```
CREATE DATABASE 1dam;
```

```
USE 1dam;
```

```
CREATE TABLE IF NOT EXISTS Herramientas ( id INT AUTO_INCREMENT PRIMARY KEY, nombre VARCHAR(50), tipo VARCHAR(50), material VARCHAR(100), uso VARCHAR(100), marca VARCHAR(50) );
```

```
INSERT INTO Herramientas (nombre, tipo, material, uso, marca) VALUES ('Martillo', 'Manual', 'Acero y madera', 'Clavar clavos', 'Stanley'), ('Destornillador', 'Manual', 'Acero y plástico', 'Atornillar/desatornillar', 'Bosch'), ('Taladro', 'Eléctrico', 'Acero y plástico', 'Perforar', 'Black & Decker'), ('Llave inglesa', 'Ajustable', 'Acero inoxidable', 'Ajustar tuercas', 'Bahco'), ('Sierra de mano', 'Manual', 'Acero y madera', 'Cortar madera', 'Irwin');
```

B.3. Creación de usuario no administrador y darle permisos sobre nuestra base de datos

Crearemos un usuario que se llame “usuario” y tenga como clave “usuario”.

Aquí están los pasos:

1. Accede a MySQL como root:

```
sudo mysql -u root -p
```

2. Verificar la política de contraseñas actual:

```
SHOW VARIABLES LIKE 'validate_password%';
```

3. Cambiar la política de contraseñas:

Si deseas permitir contraseñas más simples, puedes establecer la política en `LOW`:

```
SET GLOBAL validate_password.policy=LOW;
```

Contraseñas de tan sólo 5 caracteres:

```
SET GLOBAL validate_password.length=5;
```

4. Crear el usuario:

```
CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'usuario';
```

5. Otorgar permisos sobre la base de datos 1dam y refresca los permisos:

```
GRANT ALL PRIVILEGES ON 1dam.* TO 'usuario'@'localhost';
```

```
FLUSH PRIVILEGES;
```

6. Salir de MySQL:

```
EXIT;
```

Comprueba que todo ha funcionado:

```
fusero@latitude:~$ mysql -u usuario -pusuario
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 20
Server version: 8.0.39-0ubuntu0.24.04.2 (Ubuntu)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

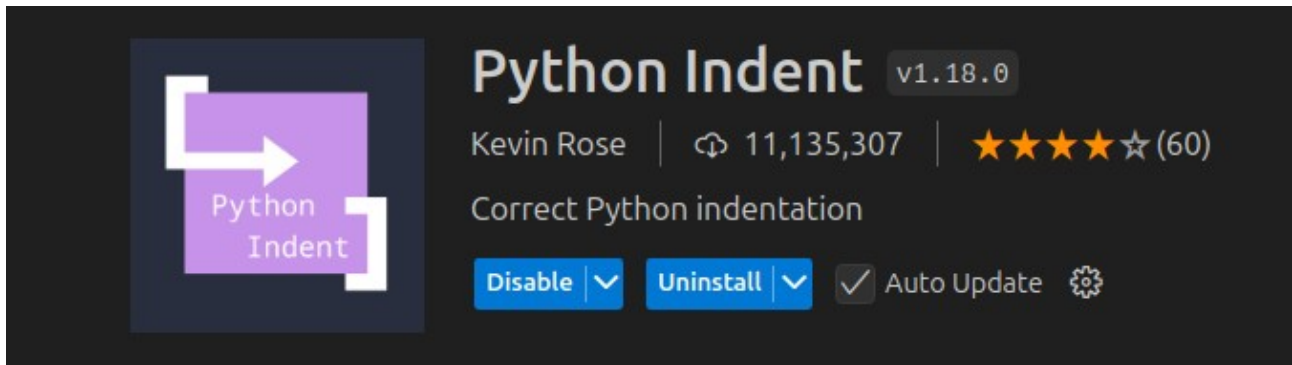
mysql> use 1dam;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from herramientas;
ERROR 1146 (42S02): Table '1dam.herramientas' doesn't exist
mysql> select * from Herramientas;
+----+-----+-----+-----+-----+-----+
| id | nombre      | tipo      | material      | uso              | marca      |
+----+-----+-----+-----+-----+-----+
| 1  | Martillo    | Manual    | Acero y madera | Clavar clavos    | Stanley    |
| 2  | Destornillador | Manual    | Acero y plástico | Atornillar/desatornillar | Bosch      |
| 3  | Taladro     | Eléctrico | Acero y plástico | Perforar         | Black & Decker |
| 4  | Llave inglesa | Ajustable | Acero inoxidable | Ajustar tuercas  | Bahco      |
| 5  | Sierra de mano | Manual    | Acero y madera | Cortar madera    | Irwin      |
+----+-----+-----+-----+-----+-----+
5 rows in set (0,00 sec)

mysql>
```

Anexo C. Configurando nuestro entorno Python

Instalamos la extensión siguiente para que gestione correctamente el indentado del código Python a medida que vayamos escribiendo nuestro script:



Instalamos el gestor de entornos virtuales:

```
sudo apt install python3-virtualenv
```

Instalamos el formateador de código:

```
sudo apt install python3-autopep8
```

