RESUMEN ACCESO A BD CON PYTHON

El manejo de bases de datos en python siempre sigue estos pasos:

1. Importar el conector
2. Conectarse a la base de datos (función connect del módulo conector)
3. Abrir un Cursor (método cursor de la conexión)
4. Ejecutar una consulta (método execute del cursor)
5. Obtener los datos (método fetch o iterar sobre el cursor)
6. Cerrar el cursor (método close del cursor)

Este es un ejemplo basico de como hacerlo con MySQL:

```
 import MySQLdb
 db = MySQLdb.connect(host="localhost", user="root",
passwd="mypassword", db="PythonU")
```

Una vez establecida la conexión, hay que crear un "cursor". Un cursor es una estructura de control que se usa para recorrer (y eventualmente procesar) los records de un result set.

El metodo para crear el cursor se llama, originalmente, cursor():

```
 cursor = db.cursor()
```

Ejemplo de ejecución de consultas:

```
import MySQLdb
db = MySQLdb.connect(host="localhost", user="root",passwd="secret",
db="PythonU")
cursor = db.cursor()
recs=cursor.execute("SELECT * FROM Students")
for x in range(recs):
   print cursor.fetchone()
```

O directamente:

```
import MySQLdb
db = MySQLdb.connect(host="localhost", user="root",passwd="secret",
db="PythonU")
cursor = db.cursor()
cursor.execute("SELECT * FROM Students")
for row in cursor:
   print row
```

1.Ejemplo por Base de Datos

SQLite

Ideal para proyectos pequeños y pruebas, ya que no requiere configuración de servidor.

```
import sqlite3

# Conexión a la base de datos (se crea si no existe)
conn = sqlite3.connect('mi_base_de_datos.db')

# Crear un cursor para ejecutar comandos SQL
cursor = conn.cursor()

# Crear una tabla
cursor.execute('''
CREATE TABLE IF NOT EXISTS usuarios (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nombre TEXT,
    edad INTEGER
)
''')

# Insertar datos
cursor.execute('INSERT INTO usuarios (nombre, edad) VALUES (?, ?)', ('Alice',
25))

# Confirmar cambios
conn.commit()

# Consultar datos
cursor.execute('SELECT * FROM usuarios')
print(cursor.fetchall())

# Cerrar conexión
conn.close()
```

MySQL

Requiere tener instalado y configurado un servidor MySQL en tu máquina local.

```
import mysql.connector

# Conexión al servidor MySQL
conn = mysql.connector.connect(
    host="localhost",
    user="tu_usuario",
    password="tu_contraseña",
    database="tu_base_de_datos"
)

# Crear un cursor
cursor = conn.cursor()
```

```
# Crear una tabla
cursor.execute('''
CREATE TABLE IF NOT EXISTS productos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50),
    precio DECIMAL(10, 2)
)
''')

# Insertar datos
cursor.execute('INSERT INTO productos (nombre, precio) VALUES (%s, %s)',
('Manzana', 1.5))

# Confirmar cambios
conn.commit()

# Consultar datos
cursor.execute('SELECT * FROM productos')
print(cursor.fetchall())

# Cerrar conexión
cursor.close()
conn.close()
```

PostgreSQL

Requiere tener PostgreSQL instalado y configurado.

```
import psycopg2

# Conexión al servidor PostgreSQL
conn = psycopg2.connect(
    host="localhost",
    user="tu_usuario",
    password="tu_contraseña",
    database="tu_base_de_datos"
)

# Crear un cursor
cursor = conn.cursor()

# Crear una tabla
cursor.execute('''
CREATE TABLE IF NOT EXISTS clientes (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(50),
    email VARCHAR(100)
)
''')

# Insertar datos
cursor.execute('INSERT INTO clientes (nombre, email) VALUES (%s, %s)', ('Juan
Pérez', 'juan@gmail.com'))

# Confirmar cambios
```

```
conn.commit()

# Consultar datos
cursor.execute('SELECT * FROM clientes')
print(cursor.fetchall())

# Cerrar conexión
cursor.close()
conn.close()
```

2.Cómo usar SQLite directamente desde Python

SQLite integrado en Python es perfecto para trabajar con bases de datos locales sin necesidad de instalar software adicional. Solo necesitas importar el módulo `sqlite3` y comenzar a trabajar.

No obstante es conveniente verificar si SQLite está instalado

1. Abre tu terminal o consola de Python.
2. Escribe el siguiente comando:

   ```
   import sqlite3
   print(sqlite3.sqlite_version)
   ```

   Si no obtienes un error y muestra la versión de SQLite, significa que ya está instalado.

SQLite viene incluido con Python. Si tienes una versión antigua de Python, actualízala a la última versión desde:

Sitio oficial de Python.

2.1. Crear o conectar una base de datos

SQLite almacenará la base de datos en un archivo `.db`. Si el archivo no existe, Python lo creará automáticamente:

```
import sqlite3

# Crear o conectar una base de datos llamada 'mi_base.db'

conn = sqlite3.connect('mi_base.db') # Crear un cursor para ejecutar comandos

SQL cursor = conn.cursor()
```

2.2. Crear una tabla

Usa el cursor para ejecutar comandos SQL y crear tablas.

```
# Crear una tabla llamada 'usuarios'
cursor.execute('''
CREATE TABLE IF NOT EXISTS usuarios (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nombre TEXT NOT NULL,
    edad INTEGER
)
''')

# Confirmar los cambios
conn.commit()
```

2.3. Insertar datos

Puedes agregar datos a la tabla con comandos SQL parametrizados para evitar inyecciones SQL:

```
python
Copiar código
# Insertar un registro en la tabla 'usuarios'
cursor.execute('INSERT INTO usuarios (nombre, edad) VALUES (?, ?)', ('Juan',
28))

# Insertar varios registros a la vez
datos = [('Ana', 24), ('Luis', 30), ('Marta', 22)]
cursor.executemany('INSERT INTO usuarios (nombre, edad) VALUES (?, ?)',
datos)

# Confirmar los cambios
conn.commit()
```

2.4. Consultar datos

Puedes recuperar datos con consultas SELECT:

```
# Consultar todos los registros
cursor.execute('SELECT * FROM usuarios')
usuarios = cursor.fetchall()

# Mostrar los resultados
for usuario in usuarios:
    print(usuario)
```

2.5. Actualizar o eliminar datos

Ejecuta comandos SQL para actualizar o eliminar registros:

```
# Actualizar un registro
cursor.execute('UPDATE usuarios SET edad = ? WHERE nombre = ?', (29, 'Juan'))

# Eliminar un registro
cursor.execute('DELETE FROM usuarios WHERE nombre = ?', ('Luis',))

# Confirmar los cambios
conn.commit()
```

2.6. Cerrar la conexión

Siempre cierra la conexión cuando termines para liberar recursos:

```
conn.close()
```

2.7. Base de datos en memoria

Si no deseas crear un archivo `.db` en el disco, puedes usar SQLite en memoria:

```
conn = sqlite3.connect(':memory:')
```

Esto crea una base de datos temporal que se elimina al cerrar la conexión.

2.8 Manejo de errores

Usa bloques `try-except` para manejar posibles errores:

```
try:
    conn = sqlite3.connect('mi_base.db')
    cursor = conn.cursor()
    # Tus operaciones aquí
except sqlite3.Error as e:
    print(f"Error: {e}")
finally:
    if conn:
        conn.close()
```