Análisis Lineal del Código: `db_manager.py`

El archivo `db_manager.py` es el guardián de la base de datos del sistema. Su única y crucial responsabilidad es gestionar todas las interacciones con la base de datos SQLite, encapsulando las consultas SQL en funciones de Python fáciles de usar. Esto abstrae la complejidad de la base de datos del resto de la aplicación, permitiendo que otros módulos, como la GUI o la máquina de estados, simplemente llamen a funciones como `agregar_usuario_bd()` sin necesidad de saber escribir SQL.

El script comienza importando las librerías necesarias: `sqlite3` es la librería estándar de Python para trabajar con bases de datos SQLite, `os` se usa para manejar rutas de archivos, y `pickle` es esencial para serializar (convertir a bytes) los "encodings" faciales, que son arrays numéricos complejos, antes de guardarlos en la base de datos.

La primera función, `inicializar_bd()`, se encarga de preparar la base de datos. Se llama al inicio del programa. Usando una consulta SQL `CREATE TABLE IF NOT EXISTS`, crea la tabla `usuarios` si no existe. La definición de la tabla es muy completa: especifica el tipo de dato de cada columna (como `TEXT` o `INTEGER`), establece restricciones como `PRIMARY KEY AUTOINCREMENT` para el ID, `UNIQUE` para el DNI y el UID RFID para evitar duplicados, y una restricción `CHECK` para asegurar que el `nivel_usuario` sea uno de los valores permitidos. Además, incluye una lógica para añadir la columna `facial_encoding` de tipo `BLOB` (Binary Large Object) si la tabla ya existía pero no tenía esta columna, lo que facilita la actualización de sistemas antiguos.

La función `agregar_usuario_bd(datos_usuario)` se utiliza para registrar un nuevo usuario. Recibe un diccionario con todos los datos del usuario. Antes de la inserción, comprueba si se proporcionó un `facial_encoding_array`. Si es así, lo serializa con `pickle.dumps()` para convertirlo en un formato binario que SQLite pueda almacenar en la columna `BLOB`. Luego, ejecuta una consulta `INSERT INTO`, pasando los datos de manera segura como parámetros (usando `?`) para prevenir inyecciones SQL. Captura errores de integridad específicos, como intentos de registrar un DNI o UID que ya existen, y devuelve mensajes de error claros.

Las funciones de obtención de datos, como `obtener_usuario_por_rfid_bd()`, `obtener_usuario_por_nombre_bd()` y `obtener_usuario_por_id_bd()`, realizan consultas `SELECT` para buscar un usuario específico. Cuando encuentran un usuario, recuperan todos sus datos. Un paso crucial aquí es que, si recuperan un `facial_encoding` de la base de datos (que está en formato de bytes), lo deserializan de nuevo a un array numérico de Python usando `pickle.loads()`. Luego, empaquetan todos los datos en un diccionario fácil de usar y lo devuelven. Si no encuentran al usuario, devuelven `None`.

Las funciones `verificar_uid_existente_bd()` y `verificar_dni_existente_bd()` son herramientas de validación. Se usan, por ejemplo, en el formulario de la GUI para comprobar si un DNI o UID ya está en uso antes de intentar guardar, permitiendo mostrar un error al usuario de forma proactiva.

La función `obtener_todos_los_usuarios_bd()` es utilizada por la GUI para poblar la tabla que muestra la lista completa de usuarios registrados.

`actualizar_usuario_bd()` se encarga de modificar los datos de un usuario existente. Construye dinámicamente una consulta `UPDATE`, añadiendo solo los campos que se van a cambiar. Esto la hace flexible y eficiente. Al igual que al agregar, serializa el `facial_encoding` si se proporciona.

Finalmente, `borrar_usuario_bd()` ejecuta una consulta `DELETE` para eliminar a un usuario de la base de datos, usando su ID como identificador único.

El bloque `if __name__ == '__main__':` permite probar el módulo de forma independiente. Ejecuta `inicializar_bd()`, intenta agregar un par de usuarios de prueba y luego realiza consultas para verificar que los datos se guardaron y se pueden recuperar correctamente. Esto es fundamental para asegurar que la capa de persistencia de datos funciona como se espera antes de integrarla con el resto del sistema.