

Bases de Datos Empotradas

Jorge L. Guillén Campos, Kendall J. Marín Muñoz, Henry D. Núñez Pérez, Isaac R.

Solís Sandí y Kun K. Zheng Liang

Instituto Tecnológico de Costa Rica, Sede Cartago

CE3101 Bases de Datos

Marco Rivera Meneses

22 de mayo del 2024

Tabla de contenidos

| | |
|-------------------------------------------------------|-----------|
| 1. Introducción..... | 2 |
| 2. Marco Teórico..... | 3 |
| ¿Qué son las bases de datos empotradas?..... | 3 |
| Ejemplos de bases de datos empotradas..... | 3 |
| Características de las Bases de Datos Empotradas..... | 4 |
| A. Integración directa..... | 4 |
| B. Eficiencia..... | 4 |
| C. Simplicidad de implementación..... | 4 |
| D. Portabilidad..... | 4 |
| E. Uso de recursos..... | 4 |
| Ventajas..... | 5 |
| A. Minimización del Consumo de Memoria..... | 5 |
| B. Reducción de Asignaciones de Recursos..... | 5 |
| C. Soporte para Múltiples Sistemas Operativos..... | 5 |
| D. Alta Disponibilidad..... | 5 |
| E. Personalización para Casos de Uso Específicos..... | 5 |
| Desventajas..... | 6 |
| A. Flexibilidad y Escalabilidad Limitadas..... | 6 |
| B. Interfaces Simplificadas..... | 6 |
| C. Compromisos en Funcionalidades Sofisticadas..... | 6 |
| D. Predictibilidad sobre Complejidad..... | 6 |
| E. Requisitos de Seguridad y Persistencia..... | 6 |
| F. Problemas Potenciales de Compatibilidad..... | 6 |
| Ámbitos de Aplicación..... | 7 |
| Comparación con Bases de Datos Independientes..... | 8 |
| 3. Desarrollo-Análisis de resultados..... | 9 |
| 4. Conclusiones..... | 17 |
| 5. Recomendaciones..... | 18 |
| 6. Bibliografía..... | 19 |

1. Introducción

En el ámbito de desarrollo de software, las bases de datos empotradas han llegado como una solución eficaz y flexible para la gestión de datos dentro de aplicaciones. Estas han existido durante varias décadas, evolucionando a lo largo del tiempo para satisfacer las necesidades cambiantes de la industria del software. Estas bases de datos, que se ejecutan directamente dentro del proceso de una aplicación, han sido una solución valiosa para gestionar los datos de manera eficiente sin la necesidad de configurar y mantener un servidor de base de datos independiente.

Desde su creación, las bases de datos empotradas han ofrecido una alternativa ligera y de alto rendimiento para aplicaciones que requieren almacenamiento local y rápido acceso a los datos. A lo largo de los años, diversas implementaciones de bases de datos empotradas han surgido, cada una con características y capacidades únicas que las hacen adecuadas para diferentes contextos y aplicaciones, proporcionando una serie de ventajas sustanciales en términos de rendimiento, simplicidad y despliegue.

El propósito de este trabajo es explorar en profundidad las bases de datos empotradas, analizando sus características, ventajas, desventajas y ámbitos de aplicación. Se presentarán ejemplos específicos de bases de datos empotradas, como SQLite, H2 y Microsoft SQL Server Compact, para ilustrar cómo estas soluciones se integran en diferentes entornos y satisfacen diversas necesidades de desarrollo.

Finalmente, se demostrará el uso práctico de una base de datos empotrada utilizando SQLite en un entorno Python. Se detalla la implementación de operaciones comunes, como la creación de tablas, la inserción de datos, la actualización y la eliminación de registros, proporcionando un ejemplo claro y funcional de cómo estas bases de datos pueden ser integradas y utilizadas eficazmente dentro de una aplicación. Este enfoque práctico permitirá visualizar las ventajas de las bases de datos empotradas en términos de eficiencia y simplicidad de uso.

2. Marco Teórico

¿Qué son las bases de datos empotradas?

Las bases de datos empotradas de acuerdo con Osorio (2019) son aquellas que no están construidas como un servicio externo, sino que están incluidas dentro del código propio de una aplicación y que pueden ser enlazadas simplemente como si se trataran de una librería. No es requerido un servidor externo para utilizarlas.

Usualmente se diferencian de las bases de datos tradicionales debido a su pequeño tamaño y la baja congestión de datos, incluso estas no se desarrollan pensando en un sistema multiusuario. Una característica primaria que resulta diferenciadora es su alta portabilidad.

Ejemplos de bases de datos empotradas

- **SQLite:** Según Rómmel (s.f.) corresponden a una herramienta *open source* que permite el almacenamiento de datos en dispositivos empotrados de forma eficiente sin limitar la potencia y opera completamente en memoria, por lo tanto, son de gran ayuda en dispositivos con baja potencia como lo son los teléfonos celulares. SQLite admite el procesamiento de consultas básicas hasta las más complejas del lenguaje SQL, lo cual en conjunto con la alta portabilidad, permite que no existan problemas de compatibilidad entre los sistemas que las utilicen, por ejemplo, entre dispositivos móviles y equipos de escritorio.
- **H2:** Para Rodríguez (2021), H2 es una base de datos relacional desarrollada en Java y que funciona enteramente en memoria. A su vez, esta posee una interfaz gráfica que se accede a través del navegador y que simula el funcionamiento de un gestor de base de datos. Dentro de sus puntos fuertes se encuentra que puede trabajar como una base embebida en aplicaciones diseñadas con Java o ser ejecutada en modo cliente-servidor, además, es de código abierto.
- **Microsoft SQL Server Compact:** De acuerdo con Microsoft (s.f.) esta base de datos es un sistema que ocupa poco espacio y permite una migración directa de datos a SQL Server, donde su sintaxis es compatible. Al igual que otras bases de datos embebidas no requiere integrarse a través de un servicio, sino que se integra en la aplicación misma. No obstante, tiene ciertas limitaciones, como por ejemplo, que tiene un tamaño máximo de 4 GB, puede haber un máximo de 1024 tablas con un máximo de 8060 bytes por registro. Es muy utilizada en entornos móviles.

Características de las Bases de Datos Empotradas

A. Integración directa

La integración directa de las bases de datos empotradas se refiere a su capacidad para ser incrustadas dentro de la aplicación. Esto significa que no debe ser ejecutado como un proceso separado, sino que se encuentra dentro del mismo proceso de la aplicación. Esto permite la reducción de latencia, facilidad de despliegue y mantenimiento simplificado.

B. Eficiencia

La eficiencia en las bases de datos empotradas se refiere a su capacidad para operar con un uso mínimo de recursos del sistema, permitiendo lograr un adecuado rendimiento para las necesidades de la aplicación. Esto permite un rendimiento óptimo, bajo consumo de energía y un tiempo de respuesta rápido.

C. Simplicidad de implementación

La simplicidad de implementación se refiere a la facilidad con la que los desarrolladores pueden integrar y usar una base de datos empotrada dentro de su aplicación. Esto implica que tiene una curva de aprendizaje reducida, menor código que gestionar y una rápida integración en el ciclo de desarrollo.

D. Portabilidad

La portabilidad de las bases de datos empotradas son una característica fundamental, ya que les permite tener funcionalidad en diferentes plataformas y entornos de desarrollo sin necesidad de modificaciones significativas. Las aplicaciones pueden ejecutarse en diversos sistemas operativos y dispositivos sin requerir cambios en la base de datos.

E. Uso de recursos

Esta característica se refiere a la manera en la cual la base de datos empotrada gestiona y utiliza los recursos del sistema como CPU, memoria y almacenamiento. Las bases de datos empotradas están diseñadas para ser eficientes en el uso de recursos. Además, utilizan menos memoria en comparación con las bases de datos tradicionales que operan como servicios independientes.

Ventajas

A. Minimización del Consumo de Memoria

Los sistemas de bases de datos incrustadas están diseñados para requerir una cantidad mínima de memoria, lo cual es crucial para los sistemas embebidos rentables. Generalmente, oscilan entre unos pocos kilobytes y un par de megabytes, haciéndolos adecuados para dispositivos con recursos de memoria limitados.

B. Reducción de Asignaciones de Recursos

Estos sistemas están optimizados para usar un ancho de banda mínimo de la CPU, permitiendo que haya más potencia de procesamiento disponible para la aplicación principal. Esta eficiencia es vital en entornos donde la base de datos y la aplicación comparten el mismo procesador.

C. Soporte para Múltiples Sistemas Operativos

Las bases de datos incrustadas son compatibles con varios sistemas operativos especializados utilizados en dispositivos embebidos, asegurando una amplia aplicabilidad en diferentes tipos de entornos de hardware y software.

D. Alta Disponibilidad

Los sistemas de bases de datos incrustadas están diseñados para operar sin supervisión administrativa continua, lo cual es esencial para sistemas donde un administrador de bases de datos no está disponible durante el tiempo de ejecución.

E. Personalización para Casos de Uso Específicos

Empresas como Pervasive ofrecen versiones de sus bases de datos adaptadas a sistemas embebidos específicos, como tarjetas inteligentes, pequeños sistemas de control y dispositivos móviles. Esta personalización asegura que el sistema de bases de datos cumpla con las necesidades particulares de cada aplicación, desde un uso mínimo de memoria hasta un avanzado control de concurrencia.

Desventajas

A. Flexibilidad y Escalabilidad Limitadas

Comparadas con las bases de datos empresariales tradicionales, las bases de datos incrustadas pueden tener una flexibilidad y escalabilidad reducidas. Están optimizadas para entornos específicos y pueden no manejar el crecimiento de datos a gran escala o transacciones complejas tan eficientemente como las bases de datos empresariales.

B. Interfaces Simplificadas

Para mejorar la velocidad y la predictibilidad, las bases de datos incrustadas a menudo presentan interfaces simplificadas. Aunque esto mejora el rendimiento, puede limitar la funcionalidad de la base de datos y hacerla menos versátil para consultas y operaciones complejas.

C. Compromisos en Funcionalidades Sofisticadas

En sistemas con memoria muy limitada, como aquellos que usan Pervasive.SQL para tarjetas inteligentes, se sacrifican características avanzadas como el control sofisticado de concurrencia y las interfaces flexibles para mantener el tamaño de la base de datos pequeño. Este compromiso puede reducir la capacidad general de la base de datos.

D. Predictibilidad sobre Complejidad

Las bases de datos incrustadas priorizan la predictibilidad y la gestión de recursos sobre la funcionalidad compleja. Esto puede ser una limitación en escenarios donde se requieren características más avanzadas de bases de datos y niveles más altos de concurrencia.

E. Requisitos de Seguridad y Persistencia

Asegurar la seguridad y la persistencia de los datos en entornos embebidos puede ser un desafío. Aunque estas bases de datos necesitan soportar diferentes niveles de seguridad y almacenamiento persistente, cumplir con estos requisitos en entornos con recursos limitados puede ser difícil.

F. Problemas Potenciales de Compatibilidad

Las bases de datos incrustadas a menudo deben soportar múltiples sistemas operativos y potencialmente interconectarse con aplicaciones externas a través de conexiones de red.

Asegurar la compatibilidad y el rendimiento en estos diversos entornos puede ser complejo y puede requerir un esfuerzo de desarrollo adicional.

Ámbitos de Aplicación

Al momento de desarrollar aplicaciones pequeñas en las que no es necesario el manejo de grandes cantidades de información no es menester el uso de un gestor de datos, por lo que cuando es así, es recomendable utilizar una base de datos embebida, que como ya se mencionó anteriormente, en ella el motor está incrustado en la aplicación. (Oscarromero, 2016)

Los usos más comunes para las bases de datos empujadas residen en aplicaciones móviles, dispositivos embebidos, aplicaciones de escritorio y algunos otros usos.

Para el caso de las *aplicaciones móviles*, las bases de datos empujadas son muy importantes para el desarrollo de estas, esto dado que permiten almacenar los datos directamente dentro del dispositivo, lo que se traduce en un mejor rendimiento y una mejora de velocidad de este. Por otro lado, este tipo de bases de datos resultan muy convenientes en respuesta de que funcionan sin la necesidad de una conexión a internet, esto evidentemente porque los datos se encuentran almacenados directamente dentro del dispositivo móvil.

Por su lado, los *dispositivos embebidos* forman parte de uno de los sectores de aplicación más comunes para las bases de datos empujadas, pues al final son indispensables para el funcionamiento de una gran variedad de sistemas, como lo puede ser el control industrial, el desarrollo de electrodomésticos inteligentes, incluso dispositivos médicos. Este género de base de datos permite realizar el seguimiento de información de sensores dentro del dispositivo, o gestionar y analizar la configuración de este, lo cual es sumamente importante.

Los dispositivos médicos de los centros sanitarios llevan tiempo incorporando sistemas integrados. “Una nueva clase de dispositivos médicos utiliza sistemas embebidos para ayudar a tratar a los pacientes que necesitan una monitorización frecuente y una atención constante en casa”

En cuanto al uso de este tipo de base de datos en *aplicaciones de escritorio*, se puede decir que las bases de datos empujadas ocupan un papel muy importante, ya que facilitan el almacenamiento de datos de usuario, además de la configuración de una posible aplicación.

Comparación con Bases de Datos Independientes

Antes mencionar que es una base de datos independiente (contained database), estas suelen referencias a una base de datos que se hospeda en un servidor independiente, por ejemplo, en el caso de las páginas web, estas son alojadas en el servidor web, esto significa que la base de datos no está directamente conectada al servidor, lo que hace a la base de datos independiente. Ahora, en cuanto a los puntos de comparación, se puede analizar el rendimiento y la dificultad de administrar este tipo de bases de datos.

Las bases de datos embebidas usualmente mejorar el funcionamiento en dispositivos con una limitada CPU o memoria (Anna, 2023), esto porque tiene una arquitectura más simple, esto a diferencia de las bases de datos servidoras o independientes, puesto que estas requieren un módulo de servidor separado, esto aumenta la complejidad y reduce un tanto el rendimiento. Dado que la comunicación entre los datos y la aplicación ocurren en los mismos procesos, suelen ser más adecuadas para alto rendimiento base de datos empotrados, especialmente en ambientes con recursos computacionales limitados. “En el espacio integrado, los recursos suelen estar restringidos y algunos dispositivos pueden depender de una batería. Por lo tanto, el rendimiento de la base de datos afecta la viabilidad de la aplicación y/o el caso de negocio. Por lo tanto, evaluar el rendimiento de la base de datos suele ser uno de los criterios más relevantes” (Anna, 2022)

Las bases de datos empotradas están directamente incrustadas en el sistema, es decir, no requieren una administración separada, por lo que se administran por sí mismas, esto evidentemente constituye en lo ventaja desde el punto de vista de la administración, pues minimiza la sobrecarga. En contraste con las bases de datos servidoras o independientes, estas generalmente requieren una administración separada y dedicada, lo que aumenta la complejidad de administración en comparación con las bases de datos empotradas.

Todas estas diferencias desembocan en la forma en que aplica y para qué se aplican cada una de este tipo de bases de datos. Las bases de datos empotradas, por sus características, generalmente se utilizan en dispositivos móviles o de consumo, en donde el rendimiento es crucial. Es por ello que estas bases se construyen directamente en la aplicación destino, favoreciendo por ejemplo, el consumo energético. No olvidar que evidentemente estas tienen limitaciones de tamaño, su objetivo no es almacenar grandes concentraciones de información. Por otro lado, las bases de datos independientes, son utilizadas en sistemas mucho más grandes, cuando factores como la escalabilidad son primordiales.

3. Desarrollo-Análisis de resultados

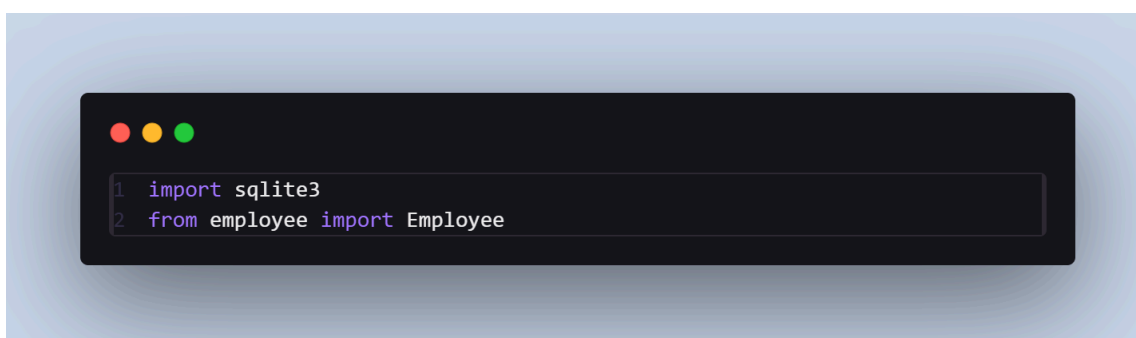
Las bases de datos empotradas de acuerdo con la información explorada permiten la portabilidad de un sistema en diferentes entornos sin la necesidad de establecer conexión con un servidor externo, no obstante, si fuese necesario estas pueden ser conectadas a un servicio externo ya sea para reportar nuevos datos o recibir información. Sin embargo, su comportamiento sigue constante en la aplicación misma, es decir, no requerirá de esa conexión externa para funcionar y mantendrá su uso local con el mismo nivel de eficiencia.

Dentro de los diferentes ejemplos explorados para bases de datos empotradas existe SQLite, la cual posee compatibilidad con diferentes equipos, tanto de escritorio como portátiles. Asimismo, SQLite puede ser utilizada en gran variedad de lenguajes de programación como C++, Java, C#, Python, entre otros. La mayoría de lenguajes que admiten SQLite tienen en común que la utilizan como una librería que puede ser importada directamente desde las configuraciones básicas del lenguajes, o bien, descargarla e importarla.

En el caso investigado se tomó como referencia el lenguaje de Python, en el cual la importación se realiza directamente tal como se observa en la Figura 1, en donde además se importa una clase externa *Employee* de la Figura 2 que permitirá posteriormente ilustrar el funcionamiento adecuado de una base de datos empotrada a través de SQLite. La implementación de esta clase cuenta con un método de inicialización que permite crear objetos que representan a un empleado con su nombre, apellido y salario.

Figura 1

Importación de la librería SQLite3 en Python



Ahora bien, una vez que se importa la librería respectiva se debe iniciar la conexión con la base de datos requerida, en este caso de acuerdo con la Figura 3, la conexión se realiza en memoria, por lo tanto está cargada en la memoria RAM y una vez que finaliza la conexión

o se cierra el programa es descartada, aunque también puede ser gestionada a través de un archivo de tal modo que los datos se preserven.

Figura 2

Implementación de una clase Employee en Python



```
1 class Employee:
2
3     def __init__(self, first, last, pay):
4         self.first = first
5         self.last = last
6         self.pay = pay
7
8     @property
9     def email(self):
10         return '{}.{}@email.com'.format(self.first, self.last)
11
12     @property
13     def fullname(self):
14         return '{} {}'.format(self.first, self.last)
15
16     def __repr__(self):
17         return "Employee('{}', '{}', {})".format(self.first, self.last,
18                                                    self.pay)
```

Figura 3

Conexión a una base de datos empotradas y la creación de una tabla para la gestión de los datos de empleados



```
1 conn = sqlite3.connect(':memory:')
2
3 c = conn.cursor()
4
5 c.execute("""CREATE TABLE employees (
6             first text,
7             last text,
8             pay integer
9         )""")
10
11 conn.close()
```

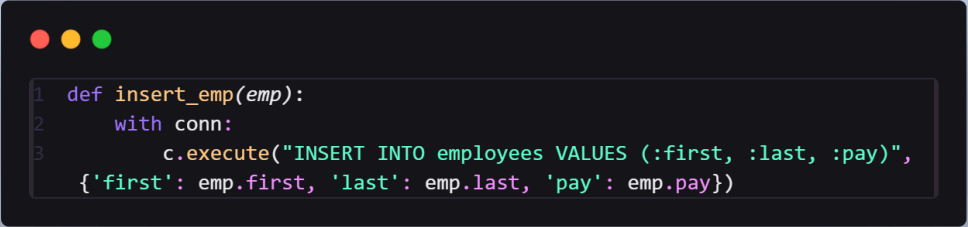
Por otro lado, en la Figura 3 también se ilustra la forma para crear una tabla en la base de datos, en donde primero deberá crear un *cursor* a partir de la conexión. Este *cursor* permite la interacción con la base de datos para realizar solicitudes, por ejemplo, la creación de la tabla *employees* mostrada, en la cual se emplea el *cursor* para ejecutar una solicitud, en este caso la creación de la tabla incluyendo sus atributos. La solicitud realizada es equivalente a la manera en la que se realizan en SQL, por lo tanto, se respalda la compatibilidad entre sistemas incluyendo servidores externos.

Un aspecto fundamental es finalizar la conexión una vez que terminen las solicitudes, de otro modo pueden ser ocasionados efectos inesperados, por ejemplo, la pérdida de datos.

En la Figura 4 se observa que la función toma un objeto *Employee* como argumento y lo inserta en la tabla *employees* empleando una sentencia SQL parametrizada para insertar los valores *first*, *last* y *pay* del empleado en la base de datos. Cabe mencionar que la conexión se maneja con una cláusula *with* asegurando que la conexión se cierre automáticamente al finalizar la operación.

Figura 4

Implementación de la función insert_emp para insertar empleados en una base de datos




```
1 def insert_emp(emp):
2     with conn:
3         c.execute("INSERT INTO employees VALUES (:first, :last, :pay)",
                    {'first': emp.first, 'last': emp.last, 'pay': emp.pay})
```

Se declaran tres objetos de la clase *employee* en la Figura 5; la información que contienen se ingresa en la tabla mediante la llamada al método *insert_emp*, que se explicó anteriormente.

Figura 5

Instanciación de objetos Employee y llamada a la función insert_emp

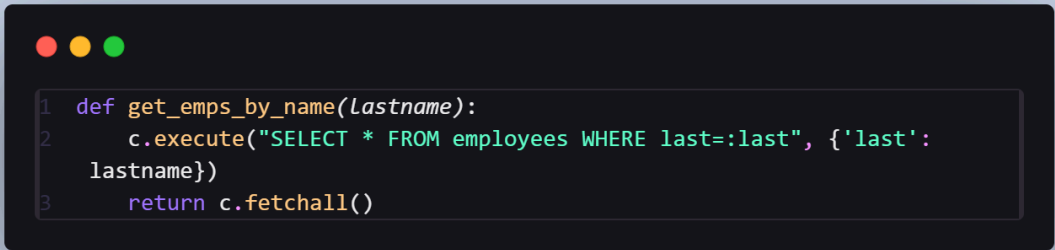


```
1 emp_1 = Employee('John', 'Doe', 80000)
2 emp_2 = Employee('Jane', 'Doe', 95000)
3 emp_3 = Employee('Marco', 'Doe', 50000)
4
5 insert_emp(emp_1)
6 insert_emp(emp_2)
7 insert_emp(emp_3)
```

Una vez se tiene poblada la tabla, para seleccionar y obtener objetos de dicha tabla, se ejecuta el query de selección, en este caso se seleccionan los objetos (personas) de la tabla *employees* que concuerden con el apellido ingresado, como se presenta en la Figura 6. Al ejecutar el método de la Figura 7 junto al parámetro “Doe”, se obtienen las tuplas donde el apellido sea el mismo al ingresado, tal como se observa en la Figura 8.

Figura 6

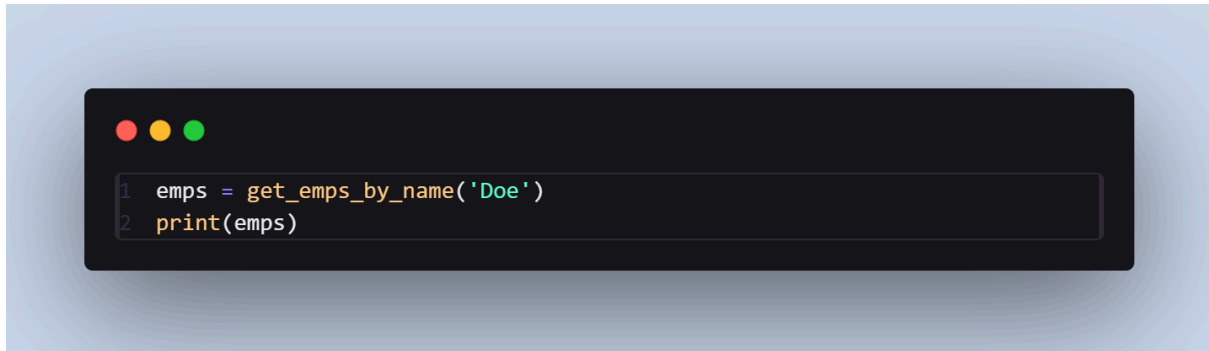
Método para selección de elementos



```
1 def get_emps_by_name(lastname):
2     c.execute("SELECT * FROM employees WHERE last=:last", {'last':
3         lastname})
4     return c.fetchall()
```

Figura 7

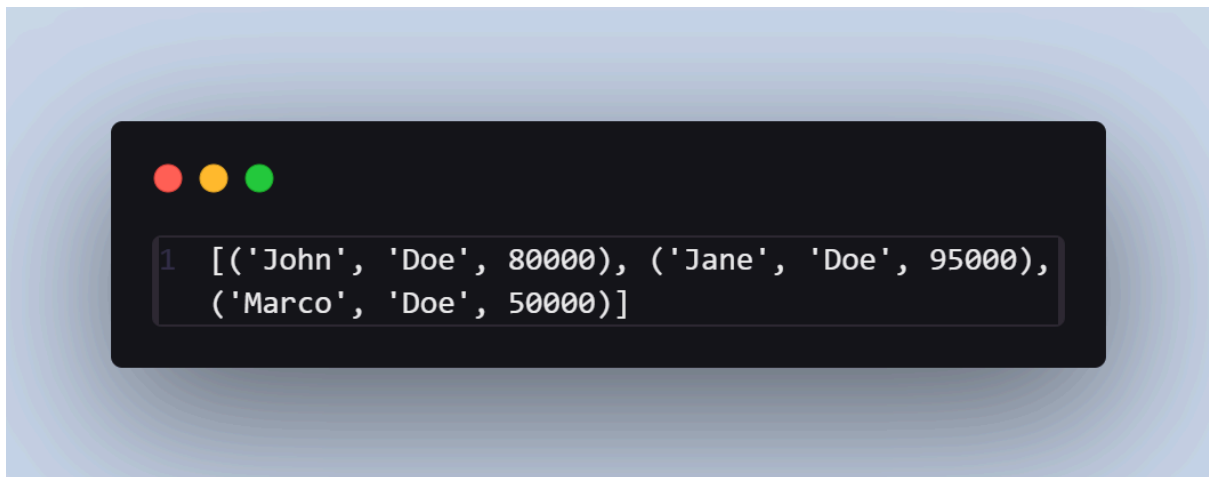
Obtención de elementos seleccionados

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains two lines of Python code:

```
1 emps = get_emps_by_name('Doe')
2 print(emps)
```

Figura 8

Impresión de empleados con apellido "Doe"

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It shows the output of the code from Figure 7 as a list of tuples:

```
1 [('John', 'Doe', 80000), ('Jane', 'Doe', 95000),
  ('Marco', 'Doe', 50000)]
```

Para actualizar una tupla de la tabla, específicamente el atributo *pay* de *employees*, se llama al método *update_pay* de la Figura 9, el cual recibe al empleado y el nuevo *pay*, como se observa en la Figura 10. En la Figura 11 se muestra en consola la actualización efectuada.

Figura 9

Método para actualizar el pay de los empleados

```
1 def update_pay(emp, pay):
2     with conn:
3         c.execute("""UPDATE employees SET pay = :pay
4                     WHERE first = :first AND last = :last""",
5                     {'first': emp.first, 'last': emp.last, 'pay': pay})
```

Figura 10

Actualización de empleados

```
1 update_pay(emp_1, 95000)
2 update_pay(emp_2, 96000)
```

Figura 11

Impresión de empleados actualizados

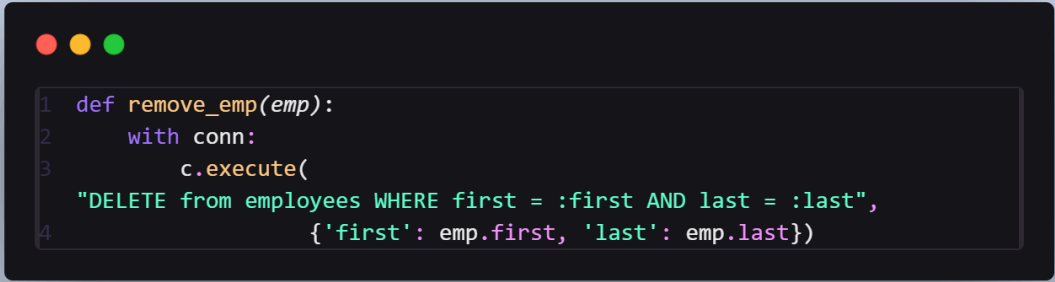
```
1 [('John', 'Doe', 80000), ('Jane', 'Doe', 95000), ('Marco', 'Doe', 5000
  0)]
2 [('John', 'Doe', 95000), ('Jane', 'Doe', 96000), ('Marco', 'Doe', 5000
  0)]
```

Cuando se desea remover un objeto perteneciente a una tabla, se ejecuta el método *remove_emp*, el cual recibe un objeto *employee* y busca en la tabla el objeto con las respectivas características para removerlo. Este método se puede observar en la Figura 12.

En la Figura 13 se llama al método *employee* junto al parámetro “emp_1” para remover este empleado de la tabla.

Figura 12

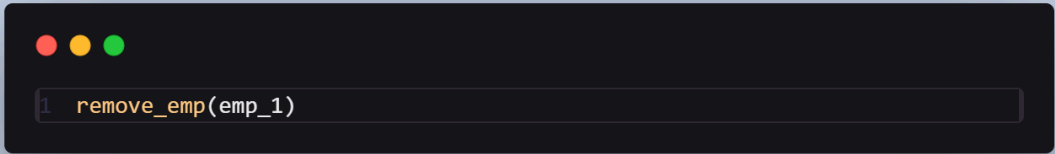
Método para remover un empleado de la tabla

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in Python and defines a function named `remove_emp` that takes an `emp` object as an argument. The function uses a `with` statement to connect to a database and then executes a `DELETE` SQL query to remove an employee from the `employees` table based on their `first` and `last` names.

```
1 def remove_emp(emp):  
2     with conn:  
3         c.execute(  
4             "DELETE from employees WHERE first = :first AND last = :last",  
             {'first': emp.first, 'last': emp.last})
```

Figura 13

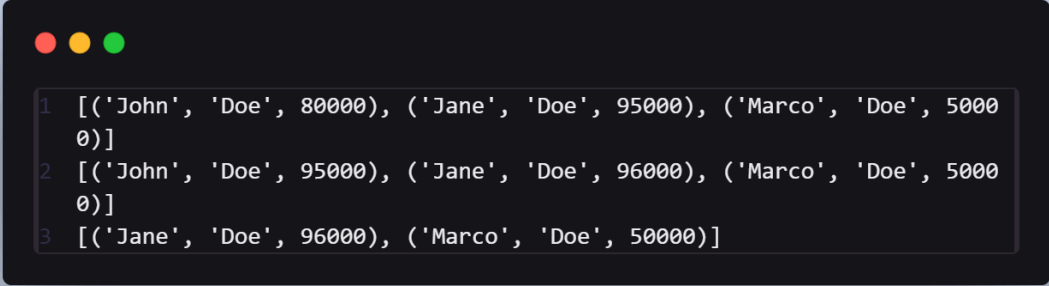
Eliminación del emp_1

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code shows a single line where the `remove_emp` function is called with the argument `emp_1`.

```
1 remove_emp(emp_1)
```


Figura 14

Impresión de empleados ejecutando el remove_emp



```
1 [('John', 'Doe', 80000), ('Jane', 'Doe', 95000), ('Marco', 'Doe', 50000)]
2 [('John', 'Doe', 95000), ('Jane', 'Doe', 96000), ('Marco', 'Doe', 50000)]
3 [('Jane', 'Doe', 96000), ('Marco', 'Doe', 50000)]
```

4. Conclusiones

Las bases de datos empotradas ofrecen una gran eficiencia y portabilidad en comparación con las bases de datos accedidas a través de un servidor, por lo tanto, permiten optimizar el funcionamiento de una aplicación y reducir su complejidad en sistemas que no ofrecen una alta potencia.

La elección entre una base de datos embebida o independiente depende de las necesidades específicas del proyecto. Para aplicaciones pequeñas con recursos limitados, es recomendable utilizar una base de datos embebida debido a su simplicidad y eficiencia. Para sistemas más grandes que requieren escalabilidad y administración centralizada, una base de datos independiente es la opción más adecuada.

Las bases de datos empotradas ofrecen una serie de ventajas significativas que las hacen ideales para una variedad de aplicaciones, especialmente las que operan en entornos con recursos limitados o que requieren una integración estrecha entre la aplicación y la base de datos. Su eficiencia, simplicidad de implementación, portabilidad y uso optimizado de recursos las convierten en una opción viable para el almacenamiento de datos ligeros.

Aunque las bases de datos embebidas enfrentan retos en la implementación de medidas de seguridad debido a las limitaciones de recursos y la falta de supervisión constante, con una personalización adecuada y una arquitectura segura, pueden proporcionar un nivel de protección suficiente para muchos entornos embebidos. No obstante, asegurar la persistencia y la integridad de los datos sigue siendo un desafío importante que requiere atención continua y soluciones específicas.

5. Recomendaciones

Es primario la utilización de bases de datos empotradas en sistemas que no son multiusuario y que no demandan una cantidad de espacio amplio, de esta forma se puede centralizar la información y mantenerla fácilmente al alcance incluso sin conexión a internet.

Se recomienda utilizar bases de datos embebidas en aplicaciones pequeñas que no requieren el manejo de grandes volúmenes de información, como aplicaciones móviles, dispositivos embebidos y aplicaciones de escritorio. Estas bases de datos ofrecen un mejor rendimiento y velocidad al almacenar los datos directamente en el dispositivo, sin necesidad de una conexión a internet, y se administran por sí mismas, lo que minimiza la complejidad de la administración.

Se recomienda utilizar las bases de datos empotradas en entornos donde la fiabilidad de los datos y el rendimiento son críticos. Además, en ocasiones en las cuales la aplicación necesite operar en condiciones de recursos limitados. Esto debido a su fiabilidad, resiliencia, rendimiento y facilidad de mantenimiento.

6. Bibliografía

¿Qué es una base de datos? (s. f.). Nutanix.

<https://www.nutanix.com/mx/info/database#:~:text=M%C3%A1s%20informaci%C3%B3n%20sobre%20NCI%20Unifique%20todas%20sus%20nubes%20Mueva%20sin%20esfuerzo%20aplicaciones%20y%20datos%20entre%20nubes%20p%C3%ABlicas%2C%20privadas%20y%20perimetrales%20para%20tener%20una%20verdadera%20experiencia%20de%20multinube%20h%C3%ADbrida>

Aaloy. (2002). *Bases de datos empotradas (I)*. BULMA.

<https://bulma.es/bases-de-datos-empotradas-i/>

Anna. (2022, 6 julio). *Embedded databases explained - Open Source by greenrobot*. Open Source By Greenrobot. <https://greenrobot.org/database/embedded-database/>

Blumenscheid, B. (s. f.). 10 Real Life Examples of Embedded Systems. *Digi*.

<https://shre.ink/82Js>

Corey Schafer. (2017, 18 abril). *Python SQLite Tutorial: Complete Overview - Creating a Database, Table, and Running Queries* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=pd-0G0MigUA>

Microsoft. (s.f.). *Microsoft® SQL Server® Compact 4.0 SP1*.

<https://www.microsoft.com/es-es/download/details.aspx?id=30709>

Osorio, J. (2019). *Bases de datos incrustadas (empotradas o embebidas)*. Technical report.

https://www.researchgate.net/publication/334067445_BASES_DE_DATOS_INCRUSTADAS_EMPOTRADAS_O_EMBEBIDAS

Rodríguez, N. (2021). *Base de Datos en memoria H2 con Spring Boot*.

REFACTORIZANDO.

<https://refactorizando.com/base-de-datos-memoria-h2-spring-boot/>

Rómmel, F. (s.f.). *SQLite: La Base de Datos Embebida*. SG.

<https://sg.com.mx/revista/17/sqlite-la-base-datos-embebida>

Tešanovic, A., Nyström, D., Hansson, J., & Norström, C. (2002). *Embedded databases for embedded real-time systems: A component-based approach*. Technical report.

<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c5a22f7f2810de00d30821f867e8b8132f07c6e5>