

Gestión de una colección con Obsidian

Publicada el 05 de enero de 2026 por Roberto Plà

No hace mucho Obsidian ha incluido una nueva prestación: la posibilidad de crear vistas de las notas similares a bases de datos. Era algo muy esperado, que de alguna manera ya estaba disponible mediante el complemento de la comunidad «dataview», pero el nuevo complemento «base» es un complemento principal que viene instalado con Obsidian.

Estas vistas, utilizan los metadatos de los archivos de la bóveda, tanto los propios del archivo: fecha de creación, nombre del archivo, fecha de modificación, etc. así como las propiedades y etiquetas que añadimos como ‘metadatos’. En realidad cada ‘base’ es una vista de los datos que Obsidian almacena sobre las notas en su propia base de datos interna.

Cuando creamos una base de datos, su contenido por defecto son todas las notas de la bóveda y tenemos que introducir ‘filtros’ para reducir los elementos a aquellos que cumplen las condiciones que imponemos, hasta formar el conjunto de notas que nos interesa. Si tenemos un proyecto o un área muy concreta, como una colección, lo único que tenemos que añadir a todas las notas o ‘fichas’ de los elementos de la colección es una propiedad característica que las distinga de las demás y así no necesitaremos más que una condición para reunir las en una vista. Para este ejemplo usaré una propiedad denominada «tiponota» a la que le daré el valor «ficha_pluma».

Estas propiedades se añaden al principio del archivo en formato YALM, y permiten tanto acotar la base de datos inicial como proporcionar los datos tabulados, «campos» o «columnas» que veremos en la vista.



Como cada registro es una nota, además de estos campos, en el cuerpo, debajo

de los datos, se puede añadir texto o cualquier otro elemento propio de una nota en Markdown, complementando los datos de las propiedades.

Para hacerse con el tema y coger soltura, hay muchos aspectos a explorar y la mejor forma de hacerlo es practicar. Así que decidí hacer una base de datos de mis plumas estilográficas. Tengo 98, y me parece un número suficiente para que el ejemplo no sea trivial. Por otra parte esta colección tiene muchos aspectos que puede ser interesante controlar. Además de marca, modelo, color y plumín, puede ser interesante conocer si historial de problemas o averías, si está, o no, entintada y con que tinta, donde está guardada, ... Suficientes aspectos para explorar todas las posibilidades del nuevo recurso.

El primer problema que hay que resolver para crear una base de datos es la definición de los campos. ¿Que características quiero tabular?. En principio los datos básicos de una pluma son su marca, modelo, color y plumín, pero también puede ser interesante la fecha de adquisición, el precio, que tipo de sistema de carga de tinta usa, si está entintada y con qué tinta. Para mi también resulta importante donde está almacenada, pues no tengo todas las plumas en el mismo cajón o caja, y prefiero tener ese dato a mano que andar revolviendo en el armario. Al final me salieron veintidós campos, que incluían una referencia que identificase cada registro de forma única y una propiedad «tags» para incluir las etiquetas o «hashtags» del archivo. Cada campo está descrito en una nota incluida en el archivo descargable con el enlace al final del artículo.

El hecho de que el registro es en realidad una nota supone una ventaja ya que, además de los campos, se puede añadir cualquier tipo de comentario, enlace o vídeos incrustados y otros datos como las fechas de entintado, la tinta usada, las fechas de limpieza, así como tareas a realizar, que pueden controlarse con el complemento «task», o incluir la imagen de la pluma en la ficha. Son muchas las posibilidades que no necesitan ser incluidas en un campo de la base de datos y que pueden complementar y ampliar la información que ofrecen estos.

Una vez decidido el diseño de la base de datos, es bueno hacer una plantilla que permita disponer de esa estructura vacía cuando creemos la nota y solo tengamos que rellenarla. No obstante si tenemos que rellenar un numero considerable de registros, crearlos uno a uno puede no ser la fórmula más cómoda.

Se me ocurrió que podría rellenar los datos cómodamente utilizando Libre Office Calc y que una vez rellenos los podría guardar el archivo como CSV y hacer un pequeño programa en Python que crease un archivo por registro pasando los datos al formato YALM que usa Obsidian para las propiedades de un archivo.

Y así lo hice. En Python, el programa básico es poco complicado. Requiere la librería CSV, de Python. También existe la librería PyYAML, que permite leer y escribir archivos YAML de manera sencilla, pero no será necesaria, ya que el formato YALM es muy sencillo y no necesitaremos manipular datos, solo escribirlos en el archivo de salida con el valor del dato precedido del nombre del campo seguido de dos puntos, a razón de un campo por línea y todas las propiedades situadas al principio del archivo entre dos líneas con tres guiones

cada una. El código del programa, así como la información adicional y los ejemplos mencionados en este artículo, están incluidos en el archivo que puede descargarse con el enlace que hay al final del artículo.

Cada archivo generado debe tener un nombre. Yo he utilizado una combinación de campos, uniendo «referencia», «marca» y «modelo» separados por guiones bajos (nunca uso espacios en los nombres de archivo) y la extensión «.md». También puede incluirse el nombre del archivo como un campo más. Lo que si he incluido son dos propiedades: «tiponota» me sirve para localizar todas las fichas que quiero reunir en una vista. Para este caso he usado el valor «ficha_pluma». También he añadido la propiedad «plantilla» que se refiere a la plantilla que hay que usar para generar la lista de propiedades en un nuevo registro. En este caso el valor también es «ficha_pluma» y refiere al archivo «ficha_pluma.md» del directorio de plantillas. La ventaja de esta propiedad es que abriendo cualquier ficha, podemos saber que plantilla debemos usar para añadir datos en una nueva nota. La plantilla y un ejemplo de nota también están en el archivo descargable.

El código del programa en Python y un los archivos ODS y CSV de ejemplo se encuentran en el archivo enlazado al final del artículo. He procurado comentar el código para que se entienda lo que hace. El programa en Python básico es bastante sencillo. Hay que importar la librería CSV y la librería 'os' que permite usar funcionalidades dependientes del sistema operativo, como abrir y cerrar archivos. Los nombres del archivo CSV de entrada y el directorio de salida se escriben en el código como variables y solo hay que abrir el CSV, leerlo y guardar los nombres de los campos que están en la primera línea en la lista 'columnas', para pasar a recorrer el archivo procesando cada línea: construir el nombre del archivo, y escribir en él los datos con la sintaxis de YALM. Y así hasta el final del archivo.

Sin embargo pensé que para que el programa tuviera 'cara y ojos' era necesario pulirlo un poco. Para empezar, consideré la posibilidad de que los datos se pudieran entrar como parámetros. Sin embargo este traslado de datos en bloque no parece que vaya a ser una operación muy frecuente. El archivo CSV tiene la estructura que yo le he dado y también la forma que yo he elegido para crear el nombre de las notas, así que otro usuario que quiera usarlo lo tendrá que modificar levemente y, si lo desea, también puede modificar las variables `archivo_entrada` y `carpeta_salida`, que se declaran al principio del código.

Sin embargo, me pareció necesario hacer algunas comprobaciones sobre la existencia de los archivos y la existencia de caracteres inválidos en los nombres de archivos. Con esas modificaciones el programa funciona y cumple su cometido.

Para ejecutar el programa, según el sistema hay que usar la orden `python csv2yalm.py` en la terminal o darle permisos de ejecución y clicar sobre él. Es mejor usar la terminal para poder leer los mensajes de error o confirmación. Una vez ejecutado el programa y creadas las notas, hay que crear la base de datos en Obsidian.

Al contar con una propiedad llamada `tiponota`, las notas pueden renombrarse o

moverse a cualquier directorio dentro de la bóveda de Obsidian. Para organizar la base de notas-registros, se puede establecer como primer filtro que la propiedad `tiponota` tenga un valor concreto, por ejemplo, `ficha_pluma`, o cualquier otro nombre que se decida usar como tipo de archivo para identificar la colección.

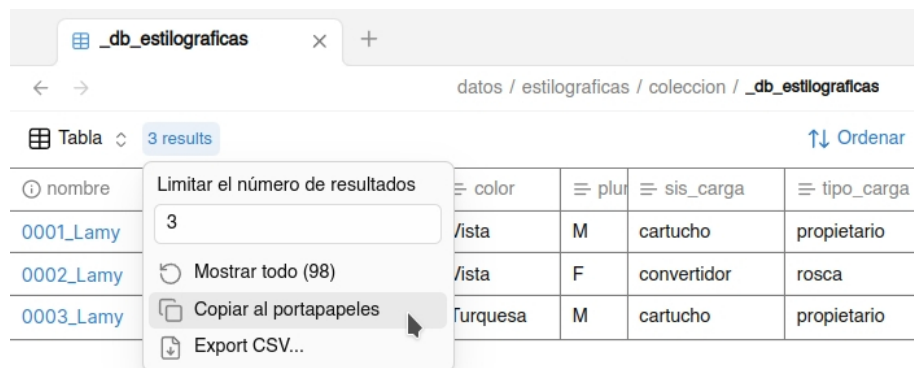
En el código Python, tanto el nombre del tipo de nota como el de la plantilla se ha definido mediante las variables `mi_tiponota` y `mi_plantilla` (líneas 21 y 22) de forma que permiten a cada usuario elegir los nombres según sus preferencias personales.

La creación de esta base de Obsidian me ha permitido ver las posibilidades de esta nueva prestación de Obsidian. Ofrece bastantes posibilidades y la hace mas competitiva ante otras opciones del mercado y permite una organizazción y un acceso a las notas dinámico, eficiente y estructurado.

Algunos de los puntos que me parecen mejorables, son estos:

A pesar de la posibilidad de usar una plantilla, que tiene que crear el usuario, para copiar en la nota-registro la estructura de campos a rellenar, la introducción de datos es incómoda. La posibilidad de rellenar los datos en el propio listado de la base de datos es un gran avance introducido con las primeras actualizaciones, pero es un tema que tiene que mejorar mucho, quizás con un complemento que permita la creación de formularios. o la importación directa desde CSV como hace mi programa.

Los tipos de datos numéricos y las fechas dan problemas de presentación. En un mismo campo puedes tener fechas o números alineados a derecha o a izquierda según Obsidian haya interpretado que esa propiedad de ese registro es de un tipo u otro. Tampoco es posible formatear la presentación de las fechas, así que una buena solución es usar todos los campos como texto, para obtener una presentación más uniforme.



The screenshot shows the Obsidian database interface. At the top, there's a tab labeled `_db_estilograficas`. Below it, a breadcrumb trail reads `datos / estilograficas / coleccion / _db_estilograficas`. A table is displayed with 3 results. The first column is labeled `nombre` and contains three entries: `0001_Lamy`, `0002_Lamy`, and `0003_Lamy`. A context menu is open over the first column, showing options: `Limitar el número de resultados` (set to 3), `Mostrar todo (98)`, `Copiar al portapapeles`, and `Export CSV...`. The table has four other columns: `color`, `plum`, `sis_carga`, and `tipo_carga`. The data rows are as follows:

nombre	color	plum	sis_carga	tipo_carga
0001_Lamy	/ista	M	cartucho	propietario
0002_Lamy	/ista	F	convertidor	rosca
0003_Lamy	turquesa	M	cartucho	propietario

La presentación de los datos de la base en Obsidian es siempre ‘virtual’, no hay un archivo que contenga la tabla que estás viendo. El archivo con extensión `«.base»`

contiene los datos o filtro para invocar la consulta y presentar las vistas. Puedes usar el portapapeles para trasladar los datos en modo texto a otro editor o en formato Markdown a una nota de Obsidian. Esto se hace con el enlace que está asociado al número de registros visibles, que aparece al lado del nombre de la vista en el lado izquierdo de la cabecera de la tabla de datos.

1 Lamy

2

3 Esta es una relación de mis plumas Lamy

4 `![[_db_lamy.base]]`

Tabla 3 results

nombre	modelo	plumin	color	precio	f_compra	entint	tinta	ubicación
0001_Lamy	Safari	M	Vista	15,5	2013	No		caja markers
0002_Lamy	Safari	F	Vista	15,5	2014	Si	Quink	escritorio
0003_Lamy	Safari	M	Turquesa	0	20190601	No		caja <u>markers</u>

5

6 Copiada y pegada como tabla Markdown:

7

8

nombre	modelo	plumin	color	precio_compra	f_compra	entintada	tinta	ubicación
0001_Lamy	Safari	M	Vista	15,5	2013	No		caja markers
0002_Lamy	Safari	F	Vista	15,5	2014	Si	Quink	escritorio
0003_Lamy	Safari	M	Turquesa	0	20190601	No		caja markers

13 Copiada y pegada como texto plano (en otra aplicación):

14

```

nombre modelo plumin color precio_compra f_compra entintada tinta ubicac:
0001_Lamy Safari M Vista 15,5 2013 No caja markers
0002_Lamy Safari F Vista 15,5 2014 Si Quink escritorio
0003_Lamy Safari M Turquesa 0 20190601 No caja markers

```

20

En el mismo menú hay una opción para exportar a CSV. Desgraciadamente esa opción no es configurable y solo exporta a archivos delimitados por comas y sin comillas en los datos tipo texto. Si usas campos con descripciones o frases has de tener cuidado de que no contengan comas. Los números con comas como separador decimal también pueden dar problemas.

En general, la exportación de documentos con datos depende la vía empleada. En la opción del menú, justo debajo de «Añadir propiedad de archivo» y encima de «Buscar» el PDF contiene los datos pero otros plugins de exportación o, por supuesto, programas externos no consiguen renderizar los datos. En HTML sale el marco y el menú de la tabla, pero no los datos. En las últimas versiones (yo uso la 1.10.6, pero hay actualizaciones anunciadas y no disponibles hasta la 1.11.3) se han incluido un gran número de mejoras y correcciones de este complemento, da la impresión de que el lanzamiento fue precipitado, casi en versión beta y es ahora cuando lo están completando aprovechando la experiencia de los usuarios. Para

exportar, lo mejor es copiar los datos a una tabla pegándolos en un documento Markdown y exportar a otro formato desde ese archivo.

La impresora tampoco es una opción viable. Obsidian no tiene una opción ‘Imprimir’ para ninguno de sus archivos supongo que la filosofía es convertir en PDF e imprimir desde ahí.

Supongo que hasta que no se establezca el desarrollo del complemento «base» no aparecerán pluguins que lo complementen. Mientras tanto siempre podemos aprovechar las opciones disponibles como la exportación a texto para programar nuestras propias opciones, como transformar el texto plano en una tabla HTML, menos costoso que «limpiar» el abultado HTML producido por los conversores.

También sería deseable poder contar con un lenguaje de consulta, una especie de SQL simple que permita recuperar datos incluso de diferentes tablas sin necesidad de construir una nueva tabla. Algo como el sistema que usa el complemento de la comunidad ‘dataview’ de Michael «Tres» Brennan «blacksmithgu». Dataview permite usar las propiedades de un archivo para filtrar las notas. Este es un ejemplo del código usado para una consulta:

```
```dataview TABLE marca, modelo, color, plumin, sis_carga
WHERE tiponota = "ficha_pluma" AND modelo="Safari" ```
```

El resultado es una tabla como puede verse en la siguiente figura. Junto al nombre del campo ‘File’ aparece el número de líneas resultado de la consulta. Aunque aparece en el mismo color que los nombres de archivo, ese número no es un enlace. Los nombres de los archivos, en la primera columna, sí son enlaces al archivo correspondiente. He incluido el path de los archivos como una columna para que pueda verse que da igual en que directorio están las notas siempre que contengan la propiedad «tiponota» que permite agruparlas.

dataview\_vista

datos / estilograficas / coleccion / dataview\_vista

dataview\_vista

1

Listado con dataview

2

3

File <sup>(6)</sup>	file.path	marca	modelo	color	plumin	sis_carga
0001_Lamy	datos/estilograficas/coleccion/salida_md/00...	Lamy	Safari	Vista	M	cartucho
0002_Lamy	datos/estilograficas/coleccion/salida_md/00...	Lamy	Safari	Vista	F	convertidor
0003_Lamy	datos/estilograficas/coleccion/salida_md/00...	Lamy	Safari	Turquesa	M	cartucho
0004_Lamy	datos/estilograficas/coleccion/salida_md/00...	Lamy	Safari	Petrol	F	cartucho
0006_Lamy	datos/estilograficas/coleccion/salida_md/00...	Lamy	Safari	Roja	M	cartucho
0007_Lamy	datos/estilograficas/mias/lamy/safari/0007_...	Lamy	Safari	Mango	F	cartucho

7

En la presentación de tabla, junto al nombre del campo 'File' aparece el número de líneas resultado de la consulta. Aunque aparece en el mismo color que los nombres de archivo, ese número no es un enlace. Los nombres de los archivos, en la primera columna, sí son enlaces al archivo correspondiente.

8

9

Código de Dataview

10

Dataview permite usar las propiedades de un archivo para filtrar las notas. Este es el código usado para la consulta anterior:

11

```

''' dataview
TABLE marca, modelo, color, plumin, sis_carga
WHERE tiponota = "ficha_pluma" AND modelo="Safari"
'''

```

12

13

14

15

16

17

La tabla que muestra dataview, puede copiarse al portapapeles y pegarla como Markdown en una nota de Obsidian donde aparece como una tabla. Con tres peculiaridades:

- Debe empezarse a resaltar el contenido a copiar desde la palabra «File». Si se empieza fuera de la tabla, lo que copia es el texto de la consulta, no su resultado.
- Debido a que el primer campo es un enlace al archivo, cuya inclusión no depende de la consulta sino que es una cuestión de diseño de 'dataview', resulta complicado empezar una selección al principio de una línea, pero puede hacerse desde cualquier punto de la línea anterior y luego eliminar esa primera línea incompleta de los datos pegados.
- Yo uso el wikienlace como formato de los enlaces, que sería algo como `[[Nombre_fichero]]`, sin embargo dataview usa el tipo de enlace clásico de Markdown : `[Nombre_fichero](app://obsidian.md/path_del_fichero/Nombre_fichero.md)` Por lo cual los enlaces en las tablas pegadas no me funcionan.

En la consulta con 'dataview' los datos aparecen en la exportación a documento HTML. Hay que decir que el documento que genera también es infumable. A partir de un fichero Markdown que pesa 732 bytes genera un HTML de 3.350.661 bytes: 783 veces más grande, donde el 0.13%, algo así como una milésima del documento es información útil. Sinceramente tardo menos en codificar a mano

la tabla HTML con un editor de texto como Kate que en limpiar el código generado.

Sin embargo tengo que añadir que estas exportaciones dependen mucho del plugin empleado. Yo tengo varios instalados y el del Menú del documento, que me aparece como «Copy as HTML» exporta la selección en una tabla mucho más simple y para HTML es el más eficaz.

Como conclusión diré que si bien la característica añadida a Obsidian no se puede llamar una «Base de Datos», ya que en realidad es una vista de la base de datos interna de Obsidian y carece de algunas de las características fundamentales de un sistema gestor de bases de datos, para tareas poco exigentes puede resultar muy útil. También cabe esperar que el equipo de programadores esté trabajando en las mejoras y pronto veamos mejoras notables.

Como suele pasar en Obsidian, el uso de los plugins adecuados cambia mucho la experiencia de usuario y en este caso complementar las bases con ‘dataview’ me parece fundamental.

El archivo baseObsidian.zip que podéis descargar, contiene el código del programa en Python, un CSV con algunos datos de ejemplos y la plantilla para generar una ficha individual. Para ver las consultas generadas con ‘dataview’ tenéis que tener instalado este plugin. Estaré encantado de leer vuestros comentarios y opiniones en Mastodón o por correo.

Esta entrada ha sido publicada en Frikilandia y etiquetada como bases-dedatos, coleccion, csv, estilografica, obsidian, programacion, python, yalm. Guarda el enlace permanente. Editar