# Taller Integrador - Python.

### Clase 7

### Sábado 09/05/2020 (Chivilcoy)

### Martes 12/05/2020 CABA.

# Bases de datos.

Como todo buen lenguaje, Python nos posibilita y facilita el trabajo con bases de datos.

Una base de datos en un conjunto de datos pertenecientes a un mismo contexto (referidas a temas relacionados), que son almacenados sistemáticamente, es decir mediante un sistema predefinido con un conjunto de reglas claras para facilitar su posterior uso y consistencia.

Generalmente se gestionan utilizando un sistema de gestión de bases de datos (en adelante SGBD).

Estos sistemas tienen sus propios lenguajes y herramientas.

### Modelos de bases de datos.

Existen varios modelos de bases de datos, dependiendo del tipo de datos e información que necesitemos almacenar.

* Jerárquicas.
* De red.
* Transaccionales.
* Relacionales.
* Documentales.
* Orientadas a objetos.
* Deductivas.

Las Jerárquicas y De red se basan en modelos del tipo árbol y establecen diferentes jerarquías a cada conjunto de datos.

Las transaccionales son bases de datos centradas en un soporte potente que permita el almacenamiento de transacciones en tiempo real.

Las relacionales y las documentales son de las más difundidas, la primera se basa en la relación existente entre los datos almacenados y las segundas, se utilizan para almacenar documentos y facilitar la búsqueda de información dentro de ellos.

Las orientadas a objetos nos permiten guardar objetos completos en la base sin necesidad de convertirlos a otro tipo de datos, como por ejemplo vimos en PHP al serializarlos.

Las deductivas permiten hacer deducciones o inferencias estadísticas sobre grandes volúmenes de datos almacenados en base a reglas específicas.

### Bases de datos relacionales.

Se basan en la idea de crear relaciones entre los diferentes conjuntos de datos, llamados entidades. Estas entidades están compuestas por registros, llamados filas o Tuplas, y cada fila a su vez está compuesta por campos (columnas).

Python permite trabajar con la mayoría de los SGBD actuales, por ejemplo:

* SQL Server.
* Oracle.
* MySql.
* PostgreSQL.
* SQLite.

Para nuestro curso vamos a utilizar SQLite, un SGBD que nos permite incorporar la base de datos a nuestro programa, no requiere instalación ni configuración.

### Bases de datos con SQLite.

Antes de comenzar a trabajar, vamos a instalar una herramienta que nos será de utilidad para ver en detalle las bases de datos con las que vayamos a trabajar.

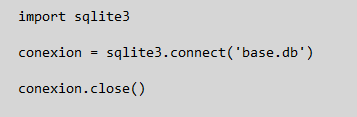
El programa se llama DB Browser for SQLite y el link de descarga es: <https://sqlitebrowser.org/>

Una vez dentro de la página, vamos a la parte de descargas y descargamos el instalador correspondiente a nuestro sistema operativo.

Lo instalamos como cualquier programa, no requiere ninguna configuración.

Al igual que sucede con los archivos de texto, no es necesario que la base de datos esté creada para utilizarla, la misma se creará al momento de hacer la conexión.

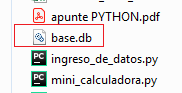
Veamos el siguiente código:



Para trabajar con bases de datos, vamos a importar el módulo sqlite3.

Luego creamos una conexión (en nuestro caso la llamamos conexión) y nos conectamos con la base de datos llamada base y cuando finalizamos de trabajar con los datos, cerramos la conexión.

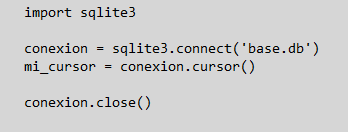
Al ejecutar el código, se nos crea la base de datos:



### Cursores y SQL.

Antes de trabajar sobre la base de datos, cosa que haremos mediante el uso de SQL, necesitamos crear un cursor.

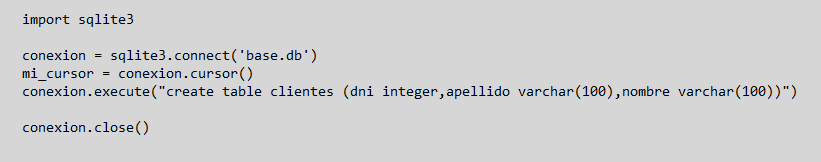
Esto lo hacemos sencillamente asignándole un nombre y utilizando la conexión que ya hemos creado:



Ahora, mediante el método EXECUTE vamos a ejecutar nuestro código en lenguaje SQL sobra la base de datos que acabamos de crear.

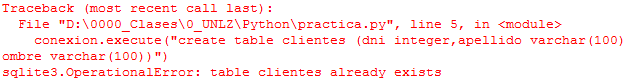
### Crear una tabla.

Vamos a crear una tabla a modo de ejemplo, para guardar los datos de nuestros clientes.



La tabla, como dijimos, la creamos utilizando SQL, y le agregamos los campos DNI, apellido y nombre del cliente.

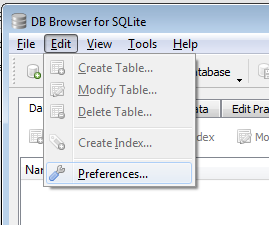
Si ejecutamos el código, se creará la nueva tabla sobre la base ya existente, y si intentamos volver a ejecutarlo, nos aparecerá el siguiente mensaje de error indicándonos que la tabla clientes que intentamos crear, ya existe:



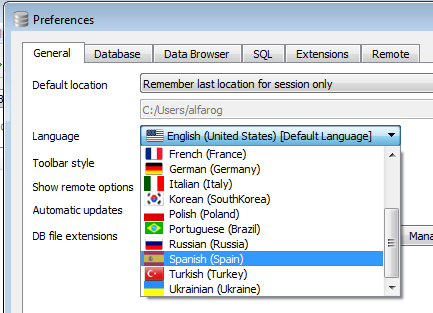
\*\*\* Ejercicio sugerido 7.1: modificar el código anterior para que no de error en caso de que la tabla ya este creada.

Veamos en detalle nuestra tabla en la base de datos, utilizando la herramienta que nos descargamos, el DB Browser.

Lo ejecutamos y si nos resulta más cómodo que esté en español, le cambiamos el lenguaje desde el menú de opciones EDIT/PREFERENCES

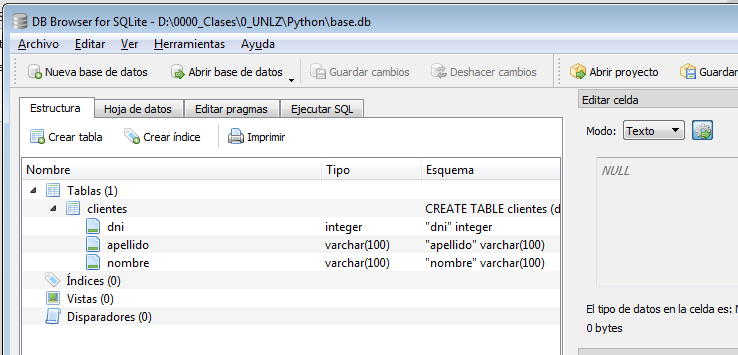


Se nos va a abrir la ventana para seleccionar las preferencias del programa, y cambiamos el idioma, luego de lo cual, grabamos los cambios:



Nos va a aparecer un mensaje avisándonos que los cambios se aplicarán una vez que reiniciemos el programa.

Una vez reiniciado vamos a abrir nuestra base de datos con la opción “Abrir base de datos” y seleccionamos la base de datos clientes que acabamos de crear:



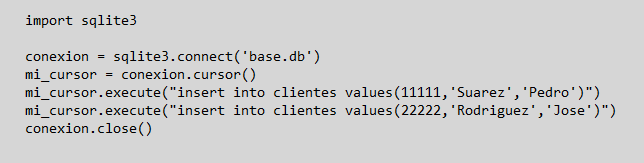
El entorno de trabajo del DB Browser es muy similar a otros SGBD, me permite trabajar sobre las bases y sus elementos, ejecutar consultas SQL y navegar los datos entre otras funcionalidades.

### Insertar datos.

Vamos a cargar un par de datos de ejemplo a nuestra tabla, mediante la sentencia INSERT de SQL.

Podemos comentar o eliminar la línea que crea la tabla para evitar el error

.

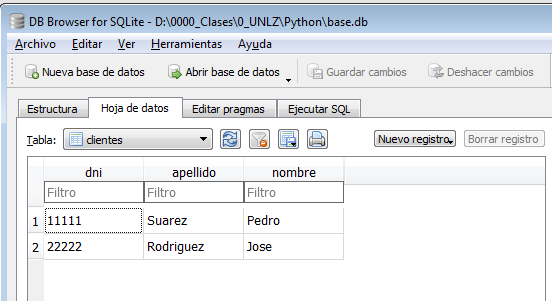


Insertamos, mediante el código anterior, un par de registros, siempre mediante el cursor.

Una vez ejecutado el código, si intentamos ver los datos desde el DB Browser veremos que la tabla continúa sin valores, y eso se debe a que luego de insertar los datos, debemos confirmar la operación mediante la sentencia COMMIT justo antes de cerrar la conexión.

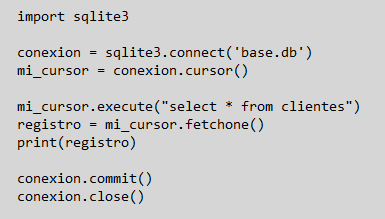


Ahora, si vamos al DB Browser, vemos los datos cargados en la tabla.



### Recuperar registros.

Veamos una manera simple de recuperar el primer registro de nuestra tabla, utilizando el método FETCHONE de nuestro cursor:



Seguimos, mediante el método EXECUTE de nuestro cursor, ejecutando el código SQL, en este caso, una consulta para recuperar todos los registros.

Si intentamos imprimir nuestro cursor, nos encontraremos con que el resultado es una referencia al objeto que no nos sirve de nada.

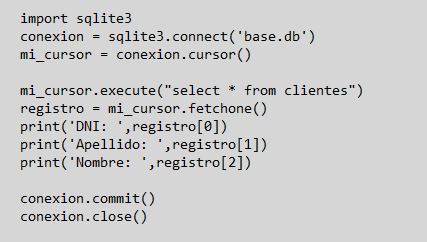
Con el método FETCHONE recuperamos el primer registro del resultado de la consulta, y lo asignamos a la variable registro, que es la que luego vamos a mostrar por pantalla:

Este es el resultado:



Si prestamos atención al resultado devuelto, vemos que es una Tupla, por lo que podremos acceder sin problemas a cada uno de sus elementos por separado.

Por ejemplo, para darle un poco de formato al resultado:



Ahora se ve así:



¿Qué sucede si volvemos a ejecutar un FETCHONE a continuación del anterior?

El puntero se mueve y nos traerá el segundo registro. Del resultado de la consulta.

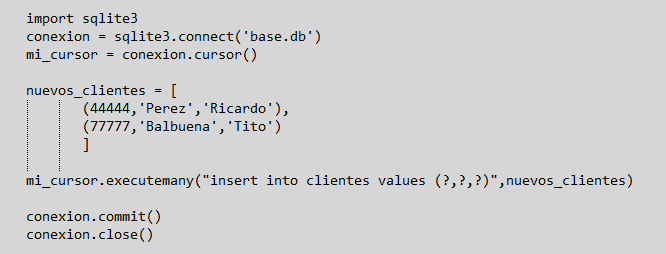
### Inserción masiva.

Tenemos un modo de cargar varios registros de una sola vez, ya sea que los tengamos en un archivo o lo tengamos cargado en memoria.

Vimos al recuperar registros, que estos vienen en forma de Tupla, y es en esa misma forma en que los vamos a cargar.

Supongamos que queremos cargar dos nuevos registros que los tenemos almacenados en forma de Tuplas.

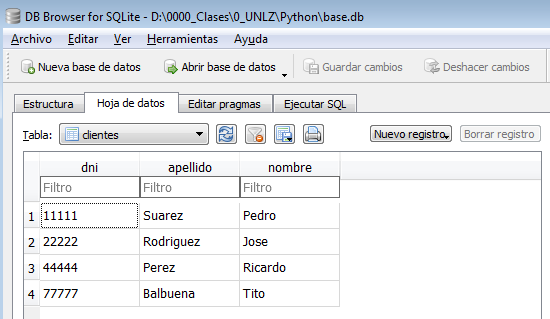
Usamos para esto, en lugar del método EXECUTE, el método EXECUTEMANY que me permite cargar una lista completa de una sola vez.



Primero cargamos en la lista “nuevos\_clientes” las Tuplas que queremos anexar a nuestra tabla.

A continuación, con EXECUTEMANY insertamos dentro de la tabla clientes, el contenido de la lista, usando la sintaxis que se ve en el ejemplo.

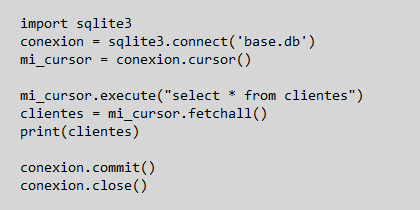
Al ejecutarlo, si actualizamos los datos de la tabla, vamos a ver cargados los que acabamos de insertar:



\*\*\* Ejercicio sugerido 7.2: escribir un programa que le pida al usuario ingresar un DNI, un Apellido y un Nombre y los guarde en la base de datos.

### Recuperación masiva.

Vimos cómo recuperar un registro de nuestra base de datos, veamos ahora como recuperar todos usando el método FETCHALL de nuestro cursor.

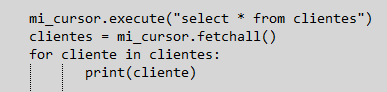


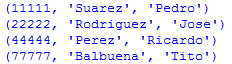
Ejecutamos un SELECT para recupera todos los registros, y con el método FETCHALL los guardamos en la variable clientes.

Cuando vemos el contenido de la variable por pantalla gracias al PRINT, lo primero que notamos es que el resultado fue devuelto en forma de lista, donde cada elemento de la lista es una tupla con los datos de cada cliente.

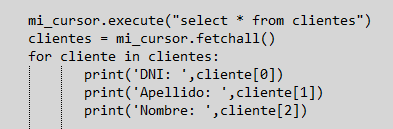


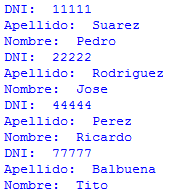
Por lo tanto, podríamos recorrerla como hacemos habitualmente cuando tenemos una lista:





Incluso podríamos darle un poco de formato mostrando cada elemento de la tupla con una leyenda:



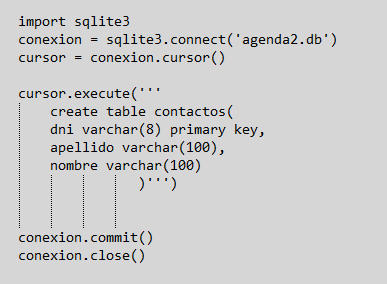


### Clave primaria.

Es un dato que actúa como identificador único de cada registro.

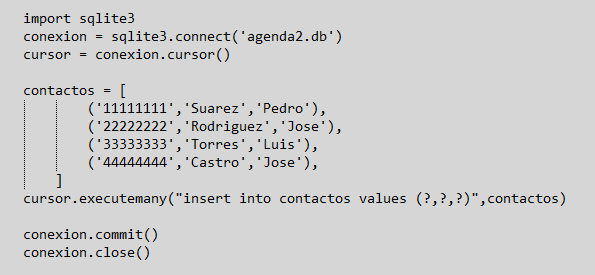
Esto quiere decir que no pueden existir dos o más registros con la misma clave primaria.

Vamos a crear una nueva base de datos llamada agenda2 y le vamos a cargar algunos datos, pero al crear la tabla, vamos a especificarle cual será la clave primaria:

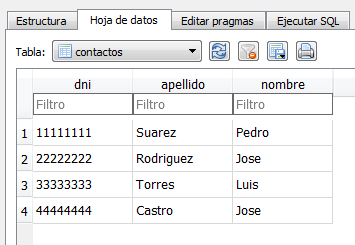


Una vez ejecutado, verificamos si existe la base y comentamos o eliminamos el código que crea la base.

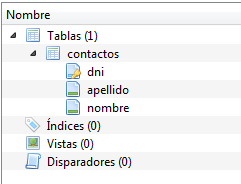
Ahora vamos a cargarle algunos datos para prueba:



Podemos verificar los datos cargados desde el DB Browser:



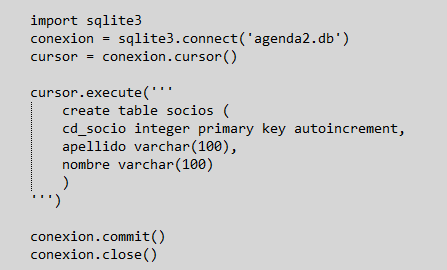
Además, si vemos la estructura de la tabla, notaremos que el campo DNI tiene una llave, indicándonos que es una clave primaria.



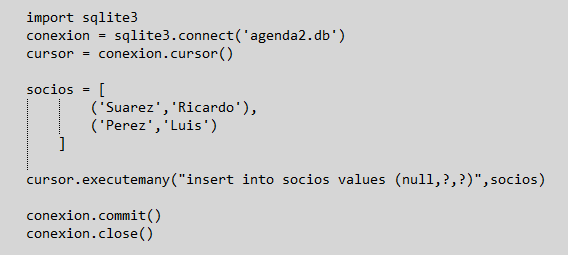
Si intentamos cargar nuevamente algún DNI que ya se encuentra en la tabla, nos encontraremos con un mensaje de error, nos dirá que lo que intentamos hacer produce un error de integridad.

### Campos auto incrementales.

Si queremos un campo numérico, que se genere solo y que además sea clave primaria, entonces debemos crear un campo auto incremental.

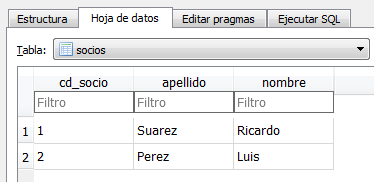


Cargamos ahora un par de valores:



Aunque no seamos nosotros quienes generemos en campo auto incremental, de todas formas, debemos declararlo en el INSERT, y lo hacemos pasando un NULL.

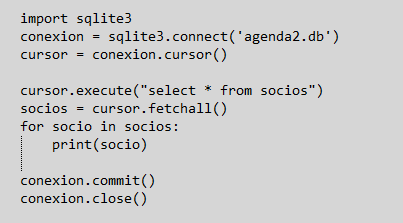
Si vemos los datos recién cargados veremos cómo se generó el campo auto incremental:

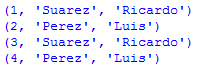


Si volvemos a ejecutar el código, volverá a cargar los mismos valores, pero el campo auto incremental será diferente, continuará aumentando.



Podemos hacer por pantalla un listado de toda la tabla para ver cómo quedó:

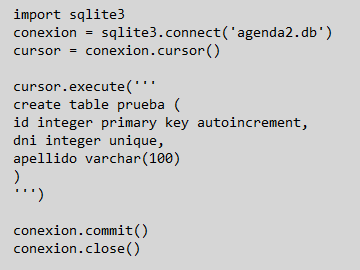




### Clave única.

El problema con las claves primarias, es que solo podemos tener una por tabla y si ponemos una clave primaria auto incremental, pero necesitamos que otro campo del registro sea único, no podemos hacerlo con otra clave primaria (nos dará error).

Para eso existen las claves únicas.



La forma de declarar una clave única es con UNIQUE.

Podemos tener más de una clave única.

### Sintaxis básica de SQL.

[Select](https://www.1keydata.com/es/sql/sql-select.php)  
SELECT columna FROM tabla

[Distinct](https://www.1keydata.com/es/sql/sql-distinct.php)  
SELECT DISTINCT columna  
FROM tabla

[Where](https://www.1keydata.com/es/sql/sql-where.php)  
SELECT columna  
FROM tabla  
WHERE condición

[And/Or](https://www.1keydata.com/es/sql/sql-and-or.php)  
SELECT columna  
FROM tabla  
WHERE condición simple

[AND|OR] condición simple

[In](https://www.1keydata.com/es/sql/sql-in.php) / Not In  
SELECT columna  
FROM tabla  
WHERE columna IN (valor1, valor2);

[Between](https://www.1keydata.com/es/sql/sql-between.php)  
SELECT columna  
FROM tabla  
WHERE columna BETWEEN valor1 AND valor2

[Like](https://www.1keydata.com/es/sql/sql-like.php)  
SELECT columna  
FROM tabla  
WHERE columna LIKE {patron}

[Order By](https://www.1keydata.com/es/sql/sql-order-by.php)  
SELECT columna  
FROM tabla   
WHERE condición  
ORDER BY columna [ASC, DESC]

[Count](https://www.1keydata.com/es/sql/sql-count.php)  
SELECT COUNT (columna)   
FROM tabla

[Group By](https://www.1keydata.com/es/sql/sql-group-by.php)  
SELECT columna1, SUM (columna2)   
FROM tabla  
GROUP BY columna1

[Having](https://www.1keydata.com/es/sql/sql-having.php)  
SELECT columna1, SUM (columna2)  
FROM tabla  
GROUP BY columna1  
HAVING (condición de función aritmética);

[Create Table](https://www.1keydata.com/es/sql/sql-create-table.php)  
CREATE TABLE tabla  
(columna1 tipo\_de\_datos\_para\_columna1,  
columna2 tipo\_de\_datos\_para\_columna2)

[Drop Table](https://www.1keydata.com/es/sql/sql-drop-table.php) (elimina los datos y la tabla)  
DROP TABLE tabla

[Truncate Table](https://www.1keydata.com/es/sql/sql-truncate-table.php) (elimina todos los datos de la tabla)  
TRUNCATE TABLE tabla

[Insert Into](https://www.1keydata.com/es/sql/sql-insert-into.php)  
INSERT INTO tabla (columna1, columna2)   
VALUES (valor1, valor2)

[Update](https://www.1keydata.com/es/sql/sql-update.php)  
UPDATE tabla  
SET columna1 = nuevo valor  
WHERE condición

[Delete From](https://www.1keydata.com/es/sql/sql-delete.php)  
DELETE FROM tabla  
WHERE condición