**SQL en Python: Preparativos**

Como comentamos en la sesión anterior podrás atacar a una base de datos SQL desde muchas plataformas/lenguajes. Por supuesto, Python es uno de ellos. Para empezar y poder acceder a nuestras primeras BBDD (Bases de datos) **utilizaremos el módulo sqlite3**.

**Configuración del entorno: Conexión**

Lo primero será importarnos sqlite3 y luego nuestro querido pandas. El objetivo de esta unidad y no sólo de esta sesión es que aprendas a extraer datos de una base de datos con SQL, llevarlos a pandas y luego todo lo demás es igual hasta lo visto hasta ahora: Análisis preliminar, duplicados, limpieza, datos faltantes, transformaciones y generación de nuevos datos que puedan servirnos y dejarlo todo preparado para el verdadero análisis o la creación de modelos.

[1]:

**import** pandas **as** pd

**import** sqlite3

Recordemos los pasos que vimos en la sesión anterior:

1. Conexión a la base de datos
2. Extracción de los datos
3. Volcado en Pandas.
4. Procesado (E)T(L) (la E la hemos hecho en 2. y la L. la veremos al final de la unidad)

Como se indica lo primero que haríamos es **establecer conexión con la base de datos** (en concreto con el gestor de bases de datos que contiene la base de datos o bases de datos a la que queramos acceder).

En esta sesión y en las siguientes vamos a leer directamente de un archivo que contiene la base de datos, pero lo normal es que tengamos que configurar la conexión a una base de datos de la empresa. En la última parte de la unidad y en algunos ejercicios sí que usaremos unas librerías de Python (pymysql) para conectarnos a un gestor MySQL externo.

Además, otras librerías y módulos que te permiten acceso a otros gestores son:

* SQL Server: pyodbc
* Oracle: cx\_Oracle
* PostgreSQL: psycopg2

[3]:

*# Conectamos con la base de datos chinook.db*

connection**=**sqlite3.connect("./data/chinook.db")

*# Obtenemos un cursor que utilizaremos para hacer las queries*

cursor\_clase**=**connection.cursor()

El cursor es un elemento común a los gestores de bases de datos y para nosotros es como un intermediario al que vamos a pasar las queries y comandos en SQL y del que obtendremos los resultados de estas queries y comandos. En general los pasos son: conexión, creación del cursor, interacción a través del cursor.

En un entorno de gestor, todo esto es transparente para el usuario que accede al intérprete de SQL (normalmente gráfico con cajitas) e interactúa directamente.

Veamos un poco más sobre el cursor y como interactuar con él

**Configuración del entorno: Cursor**

El cursor tiene varios métodos que nos interesa conocer:

* execute
* fetchall
* fetchone
* fetchmany

**execute**

El método execute es el que emplearemos para enviarle una sentencia SQL a la base de datos. A modo de ejemplo vamos a ejecutar un "SELECT \* FROM table" que recordarás que nos devolvía todas las columnas y filas de una tabla. Pero claro ¿qué tabla? En nada veremos cómo obtener las tablas que hay en una base de datos tipo Sqlite3, ahora usemos "employees" que es una tabla que tiene ese base de datos que hemos leído:

[4]:

query**=**"SELECT\*FROM employees"

cursor\_clase.execute(query)

[4]:

<sqlite3.Cursor at 0x7f5ddac53030>

Muy bien, para eso sirve execute para decirle al cursor lo que tiene que hacer, pero hasta que no usemos los otros métodos no vamos a ver nada.

**fecthone**

Este método nos devuelve el primer registro que un cursor haya obtenido al ejecutar una sentencia sql:

[5]:

cursor\_clase.fetchone()

[5]:

(1,

'Adams',

'Andrew',

'General Manager',

None,

'1962-02-18 00:00:00',

'2002-08-14 00:00:00',

'11120 Jasper Ave NW',

'Edmonton',

'AB',

'Canada',

'T5K 2N1',

'+1 (780) 428-9482',

'+1 (780) 428-3457',

'andrew@chinookcorp.com')

Devuelve una tupla

**fetchmany**

Este método nos permite recuperar un número determinado de filas, pasándole el valor por parámetro

[6]:

cursor\_clase.fetchmany(13)

[6]:

[(2,

'Edwards',

'Nancy',

'Sales Manager',

1,

'1958-12-08 00:00:00',

'2002-05-01 00:00:00',

'825 8 Ave SW',

'Calgary',

'AB',

'Canada',

'T2P 2T3',

'+1 (403) 262-3443',

'+1 (403) 262-3322',

'nancy@chinookcorp.com'),

(3,

'Peacock',

'Jane',

'Sales Support Agent',

2,

'1973-08-29 00:00:00',

'2002-04-01 00:00:00',

'1111 6 Ave SW',

'Calgary',

'AB',

'Canada',

'T2P 5M5',

'+1 (403) 262-3443',

'+1 (403) 262-6712',

'jane@chinookcorp.com'),

…

(8,

'Callahan',

'Laura',

'IT Staff',

6,

'1968-01-09 00:00:00',

'2004-03-04 00:00:00',

'923 7 ST NW',

'Lethbridge',

'AB',

'Canada',

'T1H 1Y8',

'+1 (403) 467-3351',

'+1 (403) 467-8772',

'laura@chinookcorp.com')]

Nos devuelve una lista de tuplas con los valores que le hemos pedido o el máximo de valores si este es menor.

**fetchall**

Este método nos devuelve de primeras todas las posibles filas capturadas con nuestra sentencia sql (ahora que estamos ejecutando SELECT)

[7]:

cursor\_clase.fetchall()

[7]:

[]

Sí, ha devuelto una lista vacía, porque los métodos fetch no reejecutan la query ni vuelven al principio de los resutados, cada método fetch "quita" los resultados del total, y como con fetchmany ya lo habíamos alcanzado no quedaba ninguno. Repitamos la ejecución y "fetchemos" todos los valores

[8]:

cursor\_clase.execute(query)

cursor\_clase.fetchall()

[8]:

[(1,

'Adams',

'Andrew',

'General Manager',

None,

'1962-02-18 00:00:00',

'2002-08-14 00:00:00',

'11120 Jasper Ave NW',

'Edmonton',

'AB',

'Canada',

'T5K 2N1',

'+1 (780) 428-9482',

'+1 (780) 428-3457',

'andrew@chinookcorp.com'),

…

(8,

'Callahan',

'Laura',

'IT Staff',

6,

'1968-01-09 00:00:00',

'2004-03-04 00:00:00',

'923 7 ST NW',

'Lethbridge',

'AB',

'Canada',

'T1H 1Y8',

'+1 (403) 467-3351',

'+1 (403) 467-8772',

'laura@chinookcorp.com')]

**Atributo description**

Para obtener los nombres de las columnas tenemos el atributo description.

[9]:

cursor\_clase.description

[9]:

(('EmployeeId', None, None, None, None, None, None),

('LastName', None, None, None, None, None, None),

('FirstName', None, None, None, None, None, None),

('Title', None, None, None, None, None, None),

('ReportsTo', None, None, None, None, None, None),

('BirthDate', None, None, None, None, None, None),

('HireDate', None, None, None, None, None, None),

('Address', None, None, None, None, None, None),

('City', None, None, None, None, None, None),

('State', None, None, None, None, None, None),

('Country', None, None, None, None, None, None),

('PostalCode', None, None, None, None, None, None),

('Phone', None, None, None, None, None, None),

('Fax', None, None, None, None, None, None),

('Email', None, None, None, None, None, None))

Para quedarnos con el nombre de las columnas podemos hacer algo como

[10]:

names\_columns**=**[descript[0] **for** descript **in** cursor\_clase.description]

names\_columns

[10]:

['EmployeeId',

'LastName',

'FirstName',

'Title',

'ReportsTo',

'BirthDate',

'HireDate',

'Address',

'City',

'State',

'Country',

'PostalCode',

'Phone',

'Fax',

'Email']

**Convirtiendo a pandas las salidas**

Para terminar la sesión veamos como convertir a Pandas la salida, tan sencillo como pasar la tupla obtenida y como columnas los nombres que hemos sacado de description

[13]:

cursor\_clase.execute(query)

resultado=cursor\_clase.fetchall()

df=pd.DataFrame(resultado,columns=names\_columns)

df

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente