03_Conectividad_Cursores

November 29, 2023

1 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 nuestrar primeras BBDD (Bases de datos) **utilizaremos el módulo `sqlite3`**.

1.0.1 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 las sesión anterior: 1. Conexión a la base de datos 2. Extracción de los datos con SQL 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 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 una 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`

[4]:
```
# 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()# te lo haremos impre que nos conectemos BD,␣
 ↪y es un elemento comun a todos los gestires. Siempre que accedemos a un␣
 ↪gestor de BBDD, abriremos un cursor al cual le diremos que sentencias, que␣
 ↪querys, de esta forma atacaremos esa BD, y este cursor
# lo9 alamacenenara y nosostros recuperamos la info de ahi
```

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, creacion del cursos, intereacció a través del cursor.

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

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

1.0.2 Configuración del entorno: Cursor

El cursor tiene varios métodos que nos interesa conocer:

- `execute`
- `fetchall`
- `fetchone`
- `fetcmany`

1.0.3 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 como 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:

[5]:
```
query = "SELECT * FROM employees"

cursor_clase.execute(query)# no devuelve nada de primerra hay qu eitilizar los␣
 ↪demas procedimientos
```

[5]:
```
<sqlite3.Cursor at 0x21d774d36c0>
```

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.

1.0.4 fecthone

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

[6]:
```
cursor_clase.fetchone() # los valores de la diferentes columnas en una tupla
```

[6]:
```
(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')
```

1.0.5 fetchmany

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

[7]:
```
cursor_clase.fetchmany(13) # devuelve 13 o menos de 13 si tienes menos
```

[7]:
```
[(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'),
(4,
 'Park',
 'Margaret',
 'Sales Support Agent',
 2,
 '1947-09-19 00:00:00',
 '2003-05-03 00:00:00',
 '683 10 Street SW',
 'Calgary',
 'AB',
 'Canada',
 'T2P 5G3',
 '+1 (403) 263-4423',
 '+1 (403) 263-4289',
 'margaret@chinookcorp.com'),
(5,
 'Johnson',
 'Steve',
 'Sales Support Agent',
 2,
 '1965-03-03 00:00:00',
 '2003-10-17 00:00:00',
 '7727B 41 Ave',
 'Calgary',
 'AB',
 'Canada',
 'T3B 1Y7',
 '1 (780) 836-9987',
 '1 (780) 836-9543',
 'steve@chinookcorp.com'),
(6,
 'Mitchell',
 'Michael',
 'IT Manager',
 1,
 '1973-07-01 00:00:00',
```

```
 '2003-10-17 00:00:00',
 '5827 Bowness Road NW',
 'Calgary',
 'AB',
 'Canada',
 'T3B 0C5',
 '+1 (403) 246-9887',
 '+1 (403) 246-9899',
 'michael@chinookcorp.com'),
(7,
 'King',
 'Robert',
 'IT Staff',
 6,
 '1970-05-29 00:00:00',
 '2004-01-02 00:00:00',
 '590 Columbia Boulevard West',
 'Lethbridge',
 'AB',
 'Canada',
 'T1K 5N8',
 '+1 (403) 456-9986',
 '+1 (403) 456-8485',
 'robert@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.

1.0.6 fetchall

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

[12]:
```
cursor_clase.fetchall()# no devuelve nada pq los fecth anterieres han sacado␣
↪todas las tablas existentes en el cursor. Tnedriamos qu evolver a ejecutar␣
↪la query
```

[12]:
```
[]
```

Hmmm, [Jaime que ha devuelto una lista vacía]. 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

[13]:
```
cursor_clase.execute(query)# aqui ya nos da todos
cursor_clase.fetchall()
```

[13]:
```
[(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'),
 (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'),
(4,
 'Park',
 'Margaret',
 'Sales Support Agent',
 2,
 '1947-09-19 00:00:00',
 '2003-05-03 00:00:00',
 '683 10 Street SW',
 'Calgary',
 'AB',
 'Canada',
 'T2P 5G3',
 '+1 (403) 263-4423',
 '+1 (403) 263-4289',
 'margaret@chinookcorp.com'),
(5,
 'Johnson',
 'Steve',
 'Sales Support Agent',
 2,
 '1965-03-03 00:00:00',
 '2003-10-17 00:00:00',
 '7727B 41 Ave',
 'Calgary',
 'AB',
 'Canada',
 'T3B 1Y7',
 '1 (780) 836-9987',
 '1 (780) 836-9543',
 'steve@chinookcorp.com'),
(6,
 'Mitchell',
 'Michael',
 'IT Manager',
 1,
 '1973-07-01 00:00:00',
```

```
 '2003-10-17 00:00:00',
 '5827 Bowness Road NW',
 'Calgary',
 'AB',
 'Canada',
 'T3B 0C5',
 '+1 (403) 246-9887',
 '+1 (403) 246-9899',
 'michael@chinookcorp.com'),
(7,
 'King',
 'Robert',
 'IT Staff',
 6,
 '1970-05-29 00:00:00',
 '2004-01-02 00:00:00',
 '590 Columbia Boulevard West',
 'Lethbridge',
 'AB',
 'Canada',
 'T1K 5N8',
 '+1 (403) 456-9986',
 '+1 (403) 456-8485',
 'robert@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')]
```

1.0.7 Atributo description

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

[14]:
```
cursor_clase.description#nos devlvera todas las columnas dentro de la tabla que␣
↪haya recogido el ejecutor en la sentencia(descripcitor de la columna y nos␣
↪quedamos con el 1 valor)
```

[14]:
```
(('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

[15]:
```
nombre_columnas = [descript[0] for descript  in cursor_clase.description]# una␣
 ↪lista de compresion: el for ejecuta lo que tiene delante y cada salida se va␣
 ↪acumulando como elemento la lista
nombre_columnas
```

[15]:
```
['EmployeeId',
 'LastName',
 'FirstName',
 'Title',
 'ReportsTo',
 'BirthDate',
 'HireDate',
 'Address',
 'City',
 'State',
 'Country',
 'PostalCode',
 'Phone',
 'Fax',
 'Email']
```

1.0.8 Convirtiendo a pandas las salida(un DATAFRAME DE PANDAS)

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

[17]:
```
# no necesitousar otra vez el execite pq no he ehcho otra query, si la hubiera␣
 ↪hechos hubiera yenido que ahcer el descriptor otra vez con la query nueva
cursor_clase.execute(query)
result = cursor_clase.fetchall()
```

```
df =pd.DataFrame(result, columns = nombre_columnas)
df
```

[17]:

	EmployeeId	LastName	FirstName	Title	ReportsTo	\
0	1	Adams	Andrew	General Manager	NaN	
1	2	Edwards	Nancy	Sales Manager	1.0	
2	3	Peacock	Jane	Sales Support Agent	2.0	
3	4	Park	Margaret	Sales Support Agent	2.0	
4	5	Johnson	Steve	Sales Support Agent	2.0	
5	6	Mitchell	Michael	IT Manager	1.0	
6	7	King	Robert	IT Staff	6.0	
7	8	Callahan	Laura	IT Staff	6.0	

	BirthDate	HireDate	Address	\
0	1962-02-18 00:00:00	2002-08-14 00:00:00	11120 Jasper Ave NW	
1	1958-12-08 00:00:00	2002-05-01 00:00:00	825 8 Ave SW	
2	1973-08-29 00:00:00	2002-04-01 00:00:00	1111 6 Ave SW	
3	1947-09-19 00:00:00	2003-05-03 00:00:00	683 10 Street SW	
4	1965-03-03 00:00:00	2003-10-17 00:00:00	7727B 41 Ave	
5	1973-07-01 00:00:00	2003-10-17 00:00:00	5827 Bowness Road NW	
6	1970-05-29 00:00:00	2004-01-02 00:00:00	590 Columbia Boulevard West	
7	1968-01-09 00:00:00	2004-03-04 00:00:00	923 7 ST NW	

	City	State	Country	PostalCode	Phone	Fax	\
0	Edmonton	AB	Canada	T5K 2N1	+1 (780) 428-9482	+1 (780) 428-3457	
1	Calgary	AB	Canada	T2P 2T3	+1 (403) 262-3443	+1 (403) 262-3322	
2	Calgary	AB	Canada	T2P 5M5	+1 (403) 262-3443	+1 (403) 262-6712	
3	Calgary	AB	Canada	T2P 5G3	+1 (403) 263-4423	+1 (403) 263-4289	
4	Calgary	AB	Canada	T3B 1Y7	1 (780) 836-9987	1 (780) 836-9543	
5	Calgary	AB	Canada	T3B 0C5	+1 (403) 246-9887	+1 (403) 246-9899	
6	Lethbridge	AB	Canada	T1K 5N8	+1 (403) 456-9986	+1 (403) 456-8485	
7	Lethbridge	AB	Canada	T1H 1Y8	+1 (403) 467-3351	+1 (403) 467-8772	

	Email
0	andrew@chinookcorp.com
1	nancy@chinookcorp.com
2	jane@chinookcorp.com
3	margaret@chinookcorp.com
4	steve@chinookcorp.com
5	michael@chinookcorp.com
6	robert@chinookcorp.com
7	laura@chinookcorp.com

04_Modelo_Datos_Primeras_Queries

November 29, 2023

1 SQL en Python: Primeras Queries

En la sesión anterior vimos como conectarnos a una base de datos (bueno a un fichero) y sobre todo como crear un cursor y utilizarlo para consultar la base de datos. Pero se nos quedó pendiente ver cómo podíamos saber que tablas tiene una base de datos y así ya poder empezar a trabajar de forma práctica sobre SQL. Eso vamos a hacer en esta sesión. Lo primero imports y carga la base de datos:

[1]:
```
import pandas as pd
import sqlite3

# Conectamos con la base de datos chinook.db
connection = sqlite3.connect("data/chinook.db")

# Obtenemos un cursor que utilizaremos para hacer las queries
cursor_bootcamp = connection.cursor()
```

1.0.1 Tablas y Schema

Para ver las tablas que hay en una base de datos con la que hemos establecido conexión en el caso de sqlite3:

[4]:
```

```

```
resultado = cursor_bootcamp.execute("SELECT name FROM  sqlite_master WHERE type␣
 ↪= 'table'")# hacemos un select sobre la tabla maestra(esta tabla es donde␣
 ↪estan todas las tablas de una base de datos), que tipo de datos tienen y sus␣
 ↪relaciones.
#es el catalogo de nuestra BDD. En sqlite se llama asi pero en otras BD a␣
 ↪traves de python y conseguir el nombre de las tablas maestras
tablas =[]
for name in resultado:

    print(name[0])
    tablas.append(name[0])# aqui tenemos todas las tablas
```

```
albums
sqlite_sequence
artists
customers
employees
genres
invoices
invoice_items
media_types
playlists
playlist_track
tracks
sqlite_stat1
films
```

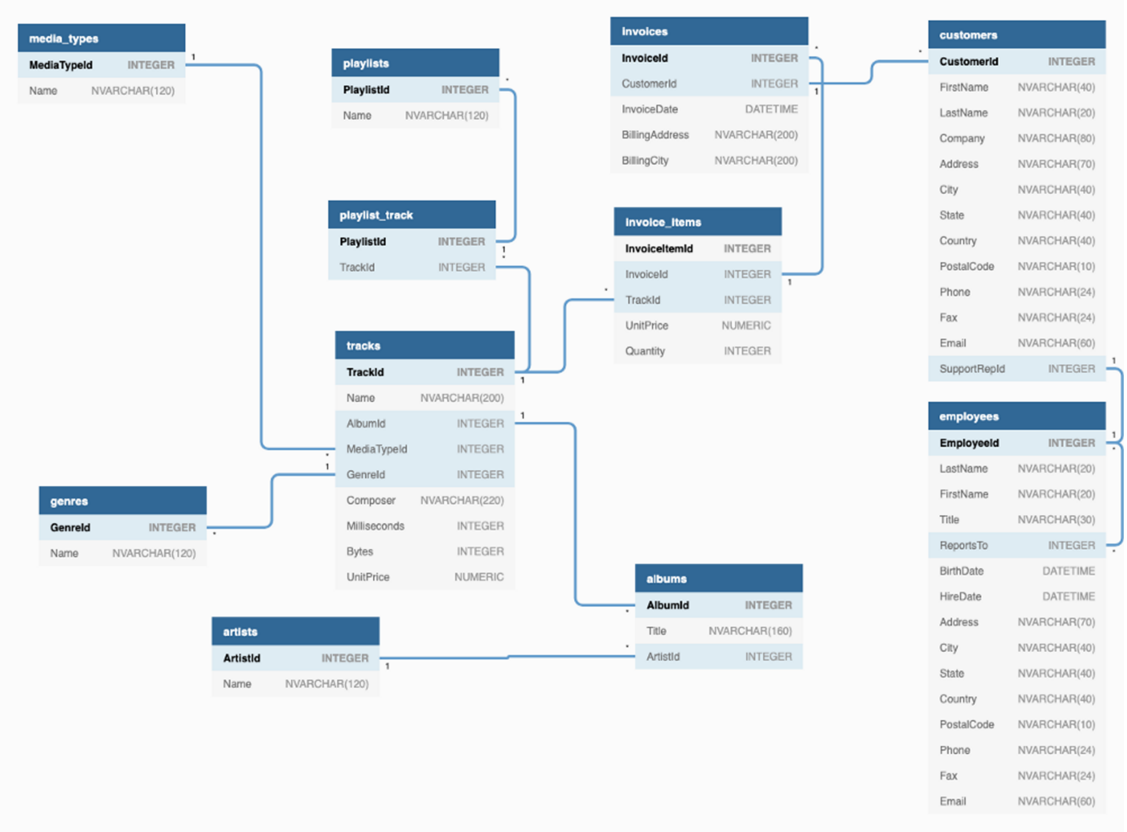
Fíjate que hemos empleado una consulta (SELECT) sql sobre una tabla, que se denomina maestra, y le hemos pasado la query al gesto a través del cursor.

Ahora podríamos recorrer todas las tablas y sacar sus nombres de columnas o podríamos investigar las columnas de esa tabla maestra y de ahí obtener el modelo de datos. Recuerda que el modelo de datos de una Base de Datos relacional es el mapa de todas las tablas de esa Base de Datos con el nombre de sus campos, el tipo de valores que guardan esos campos y las relaciones que hay entre las tablas.

1.0.2 Modelo de datos

Antes de empezar a atacar una base de datos, tendremos que saber qué hay dentro, y para ello lo mejor es ver cómo es su **modelo de datos**. Como he comentado podríamos intentar sacarlo de los nombres de los campos de las tablas o bien utilizar otros módulos externos o herramientas externas, pero dado que nosotros seremos principalmente consumidores, lo más efectivo será preguntar por él.

En nuestro caso (la base Chinook de ventas de musica onlie) este es el modelo:



1.0.3 Primeras queries con SELECT

De la sesión de introducción recordarás que la sintáxis básica de una sentencia o query SELECT tiene esta pinta:

```
SELECT campo1, campo2, campo3...
FROM tabla
WHERE condiciones
ORDER BY campo1, campo2 (DESC)
LIMIT num_filas
```

Por supuesto, hay más sentencias propias de SQL que iremos viendo a lo largo de la unidad, y del bootcamp. Fíjate que las **palabras reservadas en SQL se suelen poner en mayúsculas**, para diferenciarlas del resto. No da error si se pone de otra manera, ya que SQL **no es *case sensitive***, pero sí se suele hacer así.

En lo que queda de sesión vamos a hacer nuestras primeras queries, muy sencillas, para que vayas familiarizandote y ya en los ejercicios tendrás mucho donde practicar.

SELECT * Ya la hicimos sobre la tabla `employees` en la sesión anterior provemos ahora con otras tablas

[5]:
```
query = '''
SELECT *
FROM tracks
```

```
'''
```

Como puedes ver se suele utilizar las triple comilla y la consulta en varias líneas segúna la parte del SELECT que corresponda, pero no es imperativo, puedes poner toda la sentencia en una sóla línea tal y como hemos hecho al consultar el nombre de todas las tablas contenidas en la base de datos. Completemos la ejecución y volquemos a un `DataFrame` tal y como vimos:

[8]:
```
cursor_bootcamp.execute(query) # 1 hago el execute de la query
resultado = cursor_bootcamp.fetchall()# lo que quiero conseguir en tesecado␣
 ↪all, pero podria ser la primera o laguna many
columnas= [col[0] for col in cursor_bootcamp.description]# descriptor de las␣
 ↪columnas con una lista de progresion
df= pd.DataFrame(resultado, columns=columnas)# hago el dataframe
df
```

[8]:

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99
1	2	Balls to the Wall	2	2	1	None	342562	5510424	0.99
2	3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho…	230619	3990994	0.99
3	4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D…	252051	4331779	0.99
4	5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99
…	…	…	…	…	…	…			
3498	3499	Pini Di Roma (Pinien Von Rom) \ I Pini Della V…	343	2	24	None			
3499	3500	String Quartet No. 12 in C Minor, D. 703 "Quar…	344	2	24	Franz Schubert			
3500	3501	L'orfeo, Act 3, Sinfonia (Orchestra)	345	2	24	Claudio Monteverdi			
3501	3502	Quintet for Horn, Violin, 2 Violas, and Cello …	346	2	24	Wolfgang Amadeus Mozart			
3502	3503	Koyaanisqatsi	347	2	10	Philip Glass			

```
…          …       …         …
3498       286741  4718950      0.99
3499       139200  2283131      0.99
3500        66639  1189062      0.99
3501       221331  3665114      0.99
3502       206005  3305164      0.99

[3503 rows x 9 columns]
```

1.0.4 Selección de campos

Seleccionemos ahora algunos campos únicamente y además cambiémosle el nombre al vuelo, mediante la sintaxis `campo as nuevo_nombre`. **Si quieres poner espacios en el nombre del campo, tendrás que rodear el string con comillas dobles**

Y recuerda: SQL no es sensible a mayusculas y minusculas. ("este_CampO" == "ESTE_CAMPO")

[10]:
```
query = '''
SELECT name AS "Nombre Cancion", composer as "Compositor"
FROM tracks
'''
 # si le hubiera puesto un _ en medio de los nombres no era necesario usar ""
cursor_bootcamp.execute(query) # 1 hago el execute de la query
resultado = cursor_bootcamp.fetchall()# lo que quiero conseguir en tesecado␣
 ↪all, pero podria ser la primera o laguna many
columnas= [col[0] for col in cursor_bootcamp.description]# descriptor de las␣
 ↪columnas con una lista de progresion
df= pd.DataFrame(resultado, columns=columnas)# hago el dataframe
df
```

[10]:
```
                                      Nombre Cancion  \
0               For Those About To Rock (We Salute You)
1                                   Balls to the Wall
2                                     Fast As a Shark
3                                   Restless and Wild
4                                Princess of the Dawn
…                                                  …
3498  Pini Di Roma (Pinien Von Rom) \ I Pini Della V…
3499  String Quartet No. 12 in C Minor, D. 703 "Quar…
3500              L'orfeo, Act 3, Sinfonia (Orchestra)
3501  Quintet for Horn, Violin, 2 Violas, and Cello …
3502                                   Koyaanisqatsi

                                            Compositor
0            Angus Young, Malcolm Young, Brian Johnson
1                                                 None
2     F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho…
```

```
3     F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D…
4                              Deaffy & R.A. Smith-Diesel
…                                                     …
3498                                                 None
3499                                       Franz Schubert
3500                                    Claudio Monteverdi
3501                                 Wolfgang Amadeus Mozart
3502                                          Philip Glass

[3503 rows x 2 columns]
```

1.0.5 LIMIT y DISTINCT

Para terminar la sesión veamos como usar dos modicadores LIMIT (que ya vimos en la introducción teórica) y DISTINCT (que es el equivalete del método `unique` en pandas)

LIMIT Se usa para acotar el número de registros de la query. Va siempre al final. Por ejemplo `LIMIT 10`

[11]:
```
query = '''
SELECT Name AS "Nombre Cancion"
FROM tracks LIMIT 10'''
 # si le hubiera puesto un _ en medio de los nombres no era necesario usar ""
cursor_bootcamp.execute(query) # 1 hago el execute de la query
resultado = cursor_bootcamp.fetchall()# lo que quiero conseguir en tesecado␣
 ↪all, pero podria ser la primera o laguna many
columnas= [col[0] for col in cursor_bootcamp.description]# descriptor de las␣
 ↪columnas con una lista de progresion
df= pd.DataFrame(resultado, columns=columnas)# hago el dataframe
df
```

[11]:
```
                          Nombre Cancion
0  For Those About To Rock (We Salute You)
1                      Balls to the Wall
2                        Fast As a Shark
3                      Restless and Wild
4                   Princess of the Dawn
5                  Put The Finger On You
6                        Let's Get It Up
7                       Inject The Venom
8                             Snowballed
9                             Evil Walks
```

DISTINCT Se usa para obtener todos los registros únicos, es decir, sin duplicados. Lo podremos emplear para eliminar dupicados (aunque te recomiendo que no modifiques los datos de las bases de datos, vuelcalo a pandas y modifica ahí), como para ver todas las casuísticas de un campo en concreto.

Mucho cuidado con esta sentencia ya que si la tabla tiene miles o millones de registros, puede ralentizar mucho la query.

[12]:
```
query = '''
SELECT DISTINCT Composer ( como unique)
FROM tracks'''
 # si le hubiera puesto un _ en medio de los nombres no era necesario usar ""
cursor_bootcamp.execute(query) # 1 hago el execute de la query
resultado = cursor_bootcamp.fetchall()# lo que quiero conseguir en tesecado␣
 ↪all, pero podria ser la primera o laguna many
columnas= [col[0] for col in cursor_bootcamp.description]# descriptor de las␣
 ↪columnas con una lista de progresion
df= pd.DataFrame(resultado, columns=columnas)# hago el dataframe
df
```

[12]:
```
                                              Composer
0            Angus Young, Malcolm Young, Brian Johnson
1                                                 None
2    F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho…
3    F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D…
4                          Deaffy & R.A. Smith-Diesel
..                                                …
848                                     Carl Nielsen
849                                  Niccolò Paganini
850                           Pietro Antonio Locatelli
851                                 Claudio Monteverdi
852                                      Philip Glass

[853 rows x 1 columns]
```

[ ]:

[ ]:

05_WHERE

November 29, 2023

1 SQL en Python: WHERE

Bueno, pues ya empezamos con las clausulas WHERE aunque por ahora serán relativamente sencillas. Vamos haciendo las cargas de rigor:

[1]:
```
import pandas as pd
import sqlite3

# Conectamos con la base de datos chinook.db
connection = sqlite3.connect("data/chinook.db")

# Obtenemos un cursor que utilizaremos para hacer las queries
cursor_bootcamp = connection.cursor()
```

Pero antes de empezar...

1.0.1 Una función muy práctica

Si te fijaste en la sesión anterior repetíamos siempre el mismo código y esquema para hacer una query y luego llevarla a un dataframe. Es en esos casos en los que una función se hace necesaria... por ejemplo:

[2]:
```
# Con esta función leemos los datos y lo pasamos a un DataFrame de Pandas
def sql_query(query):

    # Ejecuta la query
```

```
    cursor_bootcamp.execute(query) # Recuerda que sólo funcionará si has␣
↪llamado cursor_bootcamp
                                   # a tu cursor, si no, cambia el nombre en␣
↪todo el código de la función

    # Almacena los datos de la query
    cursor = cursor_bootcamp.fetchall()

    # Obtenemos los nombres de las columnas de la tabla
    names = [description[0] for description in cursor_bootcamp.description]#␣
↪con los nombres que nos da el execute

    return pd.DataFrame(cursor,columns=names)
```

Probemos:

[4]:
```
query ='''
SELECT composer
FROM tracks
'''

sql_query(query)
```

[4]:
```
                                              Composer
0            Angus Young, Malcolm Young, Brian Johnson
1                                                 None
2     F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho…
3     F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D…
4                           Deaffy & R.A. Smith-Diesel
…                                                   …
3498                                              None
3499                                     Franz Schubert
3500                                  Claudio Monteverdi
3501                              Wolfgang Amadeus Mozart
3502                                       Philip Glass

[3503 rows x 1 columns]
```

1.0.2 WHERE

Se usa para filtrar filas como ya sabes, veamos algunos ejemplos de uso:

Filtros numéricos

- **Un valor numérico**
  - UnitPrice = 0.99(la comparacion aqui es con un =)
  - UnitPrice >= 0.99
  - UnitPrice < 0.99

[5]:
```
# Escogiendo la tabla tracks canciones que duren más de 6 minutos (360000␣
 ↪miliseguntos)
minutos = 6
query  = f'''
SELECT name "Nombre Cancion", Milliseconds "Duracion"
FROM tracks
WHERE Milliseconds > {minutos * 60 *1000}'''# esto no se puede hacer trabjando␣
 ↪con gestores SQL lo que padsa que aqui trabajamos con python tb

sql_query(query)
```

[5]:

	Nombre Cancion	Duracion
0	Princess of the Dawn	375418
1	Let There Be Rock	366654
2	Overdose	369319
3	Livin' On The Edge	381231
4	You Oughta Know (Alternate)	491885
..	…	…
618	Symphony No. 3 Op. 36 for Orchestra and Sopran…	567494
619	Act IV, Symphony	364296
620	3 Gymnopédies: No.1 - Lent Et Grave, No.3 - Le…	385506
621	Symphony No. 2: III. Allegro vivace	376510
622	Concerto for Violin, Strings and Continuo in G…	493573

[623 rows x 2 columns]

[8]:
```
# Escogiendo la tabla tracks canciones que duren más de 6 minutos (360000␣
 ↪miliseguntos)
minutos = 6
query  = f'''
SELECT name "Nombre Cancion", Milliseconds/1000 "Duracion"
FROM tracks
WHERE Milliseconds > {minutos * 60 *1000}'''# esto no se puede hacer trabjando␣
 ↪con gestores SQL lo que padsa que aqui trabajamos con python tb
# podemos operar los campos e el propio selct convirtiendo en minutos
sql_query(query)#
```

[8]:

	Nombre Cancion	Duracion
0	Princess of the Dawn	375
1	Let There Be Rock	366
2	Overdose	369
3	Livin' On The Edge	381
4	You Oughta Know (Alternate)	491
..	…	…
618	Symphony No. 3 Op. 36 for Orchestra and Sopran…	567
619	Act IV, Symphony	364
620	3 Gymnopédies: No.1 - Lent Et Grave, No.3 - Le…	385

```
621                Symphony No. 2: III. Allegro vivace       376
622  Concerto for Violin, Strings and Continuo in G…       493

[623 rows x 2 columns]
```

Podríamos dejarlo además en segundos operando directamente la columna en la primera parte del SELECT:

[ ]:
```
# Compositores que se llamen Brian
```

Filtros sobre campos de texto

- **Un valor string**: Name = 'Restless and Wild'
- **string contenido**:
  - strings que empiecen por 'A': Name like 'A%'
  - strings que acaben en 'A': Name like '%A'
  - strings que lleven 'A' en algun punto: Name like '%A%'

[10]:
```
query = '''
SELECT *
FROM tracks
WHERE composer LIKE "Brian%"'''

sql_query(query)
```

[10]:

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds
0	115	Please Mr. Postman	12	1	5	Brian Holland/Freddie Gorman/Georgia Dobbins/R…	137639
1	427	Who Wants To Live Forever	36	1			
2	432	Hammer To Fall	36	1			
3	1776	You've Been A Long Time Coming	146	1			
4	2125	United Colours	176	1			
5	2126	Slug	176	1			
6	2127	Your Blue Room	176	1			
7	2128	Always Forever Now	176	1			
8	2129	A Different Kind Of Blue	176	1			
9	2130	Beach Sequence	176	1			
10	2131	Miss Sarajevo	176	1			
11	2132	Ito Okashi	176	1			
12	2133	One Minute Warning	176	1			
13	2134	Corpse (These Chains Are Way Too Long)	176	1			
14	2135	Elvis Ate America	176	1			
15	2136	Plot 180	176	1			
16	2137	Theme From The Swan	176	1			
17	2138	Theme From Let's Go Native	176	1			
18	2958	Luminous Times (Hold On To Love)	234	1			

1	1		Brian May	297691
2	1		Brian May	220316
3	14	Brian Holland/Eddie Holland/Lamont Dozier		137221
4	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		330266
5	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		281469
6	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		328228
7	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		383764
8	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		120816
9	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		212297
10	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		340767
11	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		205087
12	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		279693
13	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		214909
14	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		180166
15	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		221596
16	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		203911
17	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…		186723
18	1		Brian Eno/U2	277760

	Bytes	UnitPrice
0	2206986	0.99
1	9577577	0.99
2	7255404	0.99
3	4437949	0.99
4	10939131	0.99
5	9295950	0.99
6	10867860	0.99
7	12727928	0.99
8	3884133	0.99
9	6928259	0.99
10	11064884	0.99
11	6572813	0.99
12	9335453	0.99
13	6920451	0.99
14	5851053	0.99
15	7253729	0.99
16	6638076	0.99
17	6179777	0.99
18	9015513	0.99

[11]:
```
query = '''
SELECT *
FROM tracks
WHERE composer LIKE "%Brian%"'''

sql_query(query)
```

```
[11]:    TrackId                                     Name  AlbumId  MediaTypeId  \
0          1  For Those About To Rock (We Salute You)        1            1
1          6                        Put The Finger On You        1            1
2          7                             Let's Get It Up        1            1
3          8                            Inject The Venom        1            1
4          9                                  Snowballed        1            1
5         10                                  Evil Walks        1            1
6         11                                      C.O.D.        1            1
7         12                           Breaking The Rules        1            1
8         13                      Night Of The Long Knives        1            1
9         14                                  Spellbound        1            1
10       115                          Please Mr. Postman       12            1
11       427                      Who Wants To Live Forever       36            1
12       432                               Hammer To Fall       36            1
13      1776                  You've Been A Long Time Coming      146            1
14      2125                              United Colours      176            1
15      2126                                        Slug      176            1
16      2127                              Your Blue Room      176            1
17      2128                           Always Forever Now      176            1
18      2129                     A Different Kind Of Blue      176            1
19      2130                              Beach Sequence      176            1
20      2131                               Miss Sarajevo      176            1
21      2132                                  Ito Okashi      176            1
22      2133                          One Minute Warning      176            1
23      2134        Corpse (These Chains Are Way Too Long)      176            1
24      2135                           Elvis Ate America      176            1
25      2136                                    Plot 180      176            1
26      2137                          Theme From The Swan      176            1
27      2138                     Theme From Let's Go Native      176            1
28      2257                          Fat Bottomed Girls      185            1
29      2261                                     Save Me      185            1
30      2264                                 Now I'm Here      185            1
31      2267                                       Flash      185            1
32      2269                             We Will Rock You      185            1
33      2958            Luminous Times (Hold On To Love)      234            1

    GenreId                                   Composer  Milliseconds  \
0         1          Angus Young, Malcolm Young, Brian Johnson        343719
1         1          Angus Young, Malcolm Young, Brian Johnson        205662
2         1          Angus Young, Malcolm Young, Brian Johnson        233926
3         1          Angus Young, Malcolm Young, Brian Johnson        210834
4         1          Angus Young, Malcolm Young, Brian Johnson        203102
5         1          Angus Young, Malcolm Young, Brian Johnson        263497
6         1          Angus Young, Malcolm Young, Brian Johnson        199836
7         1          Angus Young, Malcolm Young, Brian Johnson        263288
8         1          Angus Young, Malcolm Young, Brian Johnson        205688
9         1          Angus Young, Malcolm Young, Brian Johnson        270863
```

10	5	Brian Holland/Freddie Gorman/Georgia Dobbins/R…	137639
11	1	Brian May	297691
12	1	Brian May	220316
13	14	Brian Holland/Eddie Holland/Lamont Dozier	137221
14	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	330266
15	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	281469
16	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	328228
17	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	383764
18	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	120816
19	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	212297
20	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	340767
21	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	205087
22	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	279693
23	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	214909
24	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	180166
25	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	221596
26	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	203911
27	10	Brian Eno, Bono, Adam Clayton, The Edge & Larr…	186723
28	1	May, Brian	204695
29	1	May, Brian	228832
30	1	May, Brian	255346
31	1	May, Brian	168489
32	1	Deacon, John/May, Brian	122880
33	1	Brian Eno/U2	277760

	Bytes	UnitPrice
0	11170334	0.99
1	6713451	0.99
2	7636561	0.99
3	6852860	0.99
4	6599424	0.99
5	8611245	0.99
6	6566314	0.99
7	8596840	0.99
8	6706347	0.99
9	8817038	0.99
10	2206986	0.99
11	9577577	0.99
12	7255404	0.99
13	4437949	0.99
14	10939131	0.99
15	9295950	0.99
16	10867860	0.99
17	12727928	0.99
18	3884133	0.99
19	6928259	0.99
20	11064884	0.99

```
21   6572813       0.99
22   9335453       0.99
23   6920451       0.99
24   5851053       0.99
25   7253729       0.99
26   6638076       0.99
27   6179777       0.99
28   6630041       0.99
29   7444624       0.99
30   8328312       0.99
31   5464986       0.99
32   4026955       0.99
33   9015513       0.99
```

Filtros varios

- **Varios valores**: GenreId in (1, 5, 12)
- **Distinto de**: UnitPrice <> 0.99

[13]:
```
# Clientes que vivan en Berlin, Londres o París

query = '''
SELECT Firstname "Nombre", Lastname "Apellido", City "Ciudad"
FROM Customers
WHERE City in ("Berlin", "London", "Paris")'''

sql_query(query)
```

[13]:

	Nombre	Apellido	Ciudad
0	Hannah	Schneider	Berlin
1	Niklas	Schröder	Berlin
2	Camille	Bernard	Paris
3	Dominique	Lefebvre	Paris
4	Emma	Jones	London
5	Phil	Hughes	London

Combinaciones booleanas de condiciones AND Y OR Por supuesto podemos combinar los filtros ya que no dejan de ser condiciones booleanas al igual que hacemos con los `DataFrame` de pandas. Complicando un poco más el `WHERE`

[15]:
```
# Busquemos las canciones que cuesten más de 0.99 o duren menos de 3 minutos y␣
 ↪además de eso en su nombre aparezca la palabra Fire
query = '''
SELECT *
FROM tracks
WHERE (unitprice > 0.99 or Milliseconds < 3*60*1000) AND name LIKE "%fire%" '''
```

```
sql_query(query)
```

[15]:

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	1486	Fire	120	1	1	Jimi Hendrix	164989	5383075	0.99
1	1948	Fire Fire	160	1	3	Clarke/Kilmister/Taylor	164675	5416114	0.99
2	2666	Play With Fire	216	1	1	Nanker Phelge	132022	4265297	0.99
3	2892	Fire + Water	231	3	21	None	2600333	488458695	1.99
4	3181	The Fire	250	3	19	None	1288166	266856017	1.99
5	3239	Fire In Space	253	3	20	None	2926593	536784757	1.99
6	3449	Music for the Royal Fireworks, HWV351 (1749): …	315	2	24	George Frideric Handel	120000	2193734	0.99

[ ]:

06_Agregacion_Agrupacion

November 29, 2023

1 SQL con Python: Ordenación, agregación y agrupación

Vamos con una sesión con varios aspectos interesantes: ordenar la salida y agregar valores. Antes carguemos nuestros datos, librerías y la función tan util para ejecutar queries:

[2]:
```
import pandas as pd
import sqlite3

# Conectamos con la base de datos chinook.db
connection = sqlite3.connect("data/chinook.db")

# Obtenemos un cursor que utilizaremos para hacer las queries
cursor_bootcamp = connection.cursor()
```

[3]:
```
# Con esta función leemos los datos y lo pasamos a un DataFrame de Pandas
def sql_query(query):

    # Ejecuta la query
    cursor_bootcamp.execute(query) # Recuerda que sólo funcionará si has␣
 ↪llamado cursor_bootcamp
                                   # a tu cursor, si no, cambia el nombre en␣
 ↪todo el código de la función

    # Almacena los datos de la query
    ans = cursor_bootcamp.fetchall()
```

```
    # Obtenemos los nombres de las columnas de la tabla
    names = [description[0] for description in cursor_bootcamp.description]

    return pd.DataFrame(ans,columns=names)
```

1.0.1 ORDER BY

Podemos **ordenar la tabla por el campo/s que queramos**. Por defecto ordena alfabéticamente los strings y de menor a mayor los tipos numéricos. Si quieres que ordene al revés, tienes que poner `DESC` de la forma `ORDER BY campo DESC` (como el `ascending = False` del `sort_values` de pandas,o el `reverse = True` en el caso de `sort` de listas)

[5]:
```
# Obtengamos los datos de la tabla "tracks" ordenados por nombre de forma␣
 ↪descendente
query ='''' SELECT * FROM tracks ORDER BY name DESC'''

df= pd.read_sql(query, connection)

df
```

[5]:

	TrackId	Name	AlbumId	\
0	1077	Último Pau-De-Arara	85	
1	1073	Óia Eu Aqui De Novo	85	
2	2078	Óculos	169	
3	3496	Étude 1, In C Major - Preludio (Presto) - Liszt	340	
4	333	É que Nessa Encarnação Eu Nasci Manga	29	
…	…	…	…	
3498	3254	#9 Dream	255	
3499	109	#1 Zero	11	
3500	3412	"Eine Kleine Nachtmusik" Serenade In G, K. 525…	281	
3501	2918	"?"	231	
3502	3027	"40"	239	

	MediaTypeId	GenreId	Composer	Milliseconds	\
0	1	10	Corumbá/José Gumarães/Venancio	200437	
1	1	10	None	219454	
2	1	7	None	219271	
3	4	24	None	51780	
4	1	9	Lucina/Luli	196519	
…	…	…	…	…	
3498	2	9	None	278312	
3499	1	4	Cornell, Commerford, Morello, Wilk	299102	
3500	2	24	Wolfgang Amadeus Mozart	348971	
3501	3	19	None	2782333	
3502	1	1	U2	157962	

	Bytes	UnitPrice

```
0       6638563       0.99
1       7469735       0.99
2       7262419       0.99
3       2229617       0.99
4       6568081       0.99
…          …          …
3498    4506425       0.99
3499    9731988       0.99
3500    5760129       0.99
3501  528227089       1.99
3502    5251767       0.99

[3503 rows x 9 columns]
```

[19]:
```
# Obtengamos el nombre de los clientes norteamericanos ordenados por apellido

query = '''
SELECT Firstname "Nombre", Lastname "apellido", Country "Pais"
FROM Customers
WHERE Country in ("USA", "Canada")
ORDER BY LastName'''

df = pd.read_sql(query, connection)

df
```

[19]:

	Nombre	apellido	Pais
0	Julia	Barnett	USA
1	Michelle	Brooks	USA
2	Robert	Brown	Canada
3	Kathy	Chase	USA
4	Richard	Cunningham	USA
5	Edward	Francis	Canada
6	John	Gordon	USA
7	Tim	Goyer	USA
8	Patrick	Gray	USA
9	Frank	Harris	USA
10	Heather	Leacock	USA
11	Dan	Miller	USA
12	Aaron	Mitchell	Canada
13	Jennifer	Peterson	Canada
14	Mark	Philips	Canada
15	Frank	Ralston	USA
16	Martha	Silk	Canada
17	Jack	Smith	USA
18	Victor	Stevens	USA
19	Ellie	Sullivan	Canada

```
20  François    Tremblay  Canada
```

1.0.2 Agregaciones

Como ocurre cuando trabajamos con datos en ocasiones nos interesa obtener algún estadístico como el máximo de un campo, su desviación estándar o simplemente un conteo de registros no nulos. Para ello podemos usar funciones como `MAX`, `COUNT` o `AVG`. En esta página encontrarás un resumen con las principales funciones.

[21]:
```
# Obtener el número de canciones que contienen la palabra love
query = ''' SELECT COUNT (*) FROM tracks WHERE  name LIKE "%love%"'''
df =pd.read_sql(query, connection)
df
```

[21]:
```
   COUNT (*)
0        114
```

[22]:
```
# Encontrar la media del precio de las canciones compradas
query = '''SELECT AVG(unitprice) FROM invoice_items'''
df =pd.read_sql(query, connection)
df
```

[22]:
```
   AVG(unitprice)
0        1.039554
```

[24]:
```
# Obtener la duración máxima de las canciones
query ='''SELECT MAX(Milliseconds)/1000 FROM tracks'''
df =pd.read_sql(query, connection)
df
```

[24]:
```
   MAX(Milliseconds)/1000
0                    5286
```

1.0.3 Agrupaciones (GROUP BY):

Para terminar veamos como se hacen agrupaciones empleando GROUP BY, y de forma muy similar a como se hace con pandas.

[25]:
```
# Selección del precio unitario en funcion del género
query = ''' SELECT GenreId, SUM(unitprice) as TOT_PRICE FROM tracks GROUP BY␣
 ↪GenreId ORDER BY TOT_PRICE DESC LIMIT 10'''
df =pd.read_sql(query, connection)
df
```

[25]:
```
   GenreId  TOT_PRICE
0        1    1284.03
1        7     573.21
2        3     370.26
```

```
3        4     328.68
4       19     185.07
5        2     128.70
6       21     127.36
7        6      80.19
8       24      73.26
9       14      60.39
```

O calcular cuantas canciones hay por compositor

[26]:
```
query = ''' SELECT composer , COUNT(trackId) FROM tracks WHERE Composer IS NOT␣
 ↪NULL -- COMO ISNA EN PANDAS
GROUP BY composer
ORDER BY COUNT(trackId)-- PODRIA PONER UN 2 PQ ESTA EN 2 LUGAR  EN EL SELCT'''
df =pd.read_sql(query, connection)
df
```

[26]:

	Composer	COUNT(trackId)
0	A. Jamal	1
1	A.Bouchard/J.Bouchard/S.Pearlman	1
2	A.Isbell/A.Jones/O.Redding	1
3	Aaron Copland	1
4	Aaron Goldberg	1
..	…	…
847	Kurt Cobain	26
848	Billy Corgan	31
849	Jagger/Richards	35
850	U2	44
851	Steve Harris	80

[852 rows x 2 columns]

Podrás practicar con todo lo visto en los ejercicios correspondientes, en los que además practicarás con la combinación de queries y subqueries.

[ ]:

[ ]:

07_Joins_Teoria

November 29, 2023

1 SQL con Python: JOIN

Hasta ahora hemos hecho queries sobre una única tabla, pero **¿y si queremos juntar datos de varias tablas?** Para eso están los `JOIN`s. Para ello **necesitas tener uno o varios campos comunes entre ambas tablas, que se denominan CLAVES**. [como ya vimos en la lejana sesión inicial dedicada a bases de datos relacionales]

¿Cuándo usarlos? Por ejemplo, si tenemos una tabla con un conjunto de clientes y necesitamos añadirles campos nuevos, tendremos que acudir a otras tablas donde esté ese identificador de cliente y aplicar un `JOIN`. Es lo que se conoce como *pegar campos* a otra tabla a traves de esos JOINS

O imagina que tienes una tabla con todos tus pedidos, con muchos campos(ciudad, dirección, cliente...) y en otra tabla únicamente los números de pedido que no se llegaron a entregar. Si quieres filtrar dentro de tu tabla total de pedidos los que no se llegaron a entregar, podrías aplicar lo que se llama un `INNER JOIN` de manera que te quedes con lo común en ambas tablas, siendo tu clave el identificativo del pedido...

Veamos la teoría sobre los JOIN que existe, pero no te agobies con los nombres, no necesitas saber si estás haciend in inner, un outter o un left inner outter simpatetic join, lo que necesitas es saber como quieres juntar tus tablas y luego aplicar el concepto adecuado.

1.0.1 Tipos de JOINS

Vamos a verlo primero con un ejemplo sencillo de dos tablas:

Tabla 1: Empleados

id_empleado	nombre	id_departamento
1	Ana	10
2	Carlos	20
3	Diana	10
4	Eduardo	30
5	Luis	-1

(El -1 quiere decir por ejemplo que todavía no tiene departamento asignado)

Tabla 2: Departamentos

id_departamento	nombre_departamento
10	Marketing
20	Ventas
30	IT
40	Recursos Humanos

Como puedes ver tienen dos columnas que comparten valores y que (en este caso) se llaman igual: `id_departamento`

Supongamos ahora que queremos unirlas usando esa columna común y que llamamos tabla de la izquierda a la tabla 1 y tabla de la derecha a la tabla_2, cuántas formas hay de hacerlo...:(TOD O JOIN TIENE UNA PARTE IZQ Y OTRA DEREHC)

1.0.2 Left (Outer) Join

busca para cada elemento existente en la tabla de la izq, cuale sson los elementos segun esa clave de uncion y les pega los datos , se llama left join

1. Podemos hacerlo considerando únicamente la tabla_1 como directora es decir unimos a cada empleado los datos del departamento que corresponda con el id_departamento. El resultado sería:

id_empleado	nombre	id_departamento	nombre_departamento
1	Ana	10	Marketing
2	Carlos	20	Ventas
3	Diana	10	Marketing
4	Eduardo	30	IT
5	Luis	-1	NaN/Null

como luis -1 no exste deprtamento le asigna un NULL/NaN

Hemos pegado el nombre del departamento correspondiente y a Luis un NaN o Null porque su departamento no existe en la tabla 2.

Bien a este JOIN en el que manda la tabla de la IZQUIERDA o primera tabla mencioanda se le llama LEFT JOIN (que curioso verdad) y también LEFT OUTER JOIN.

Su sintaxis en SLQ sería algo como:

```
SELECT A.id_empleado, A.nombre, A.id_departamento, B.nombre_departamento # las tablas le puedes
FROM Empleados AS A( la tabla que viene detras del FROM es la tabla que manda y sera seimpre
LEFT JOIN Departamentos AS B
ON A.id_departamento = B.id_departamento
```

Lo primero que te llamará la atención son los alias es decir eso que aparece como `as A` o `as B` detrás del nombre de las tablas. Sí podemos ponerle alias a las tablas para luego referirnos a sus campos con la **notación <alias>.<campo>**. Ojo podemos prescindir de los alias y **usar <nombre_tabla>.<campo>** pero cuando los nombres son largos se suelen alias.

Superado el tema del alias, ves que la tabla de la izquierda es la primera que aparece (la que está detrás del `FROM`) y la de la derecha la que aparece en tras la palabra `JOIN`.

Finalmente se usa `ON` para anticipar la parte de las claves que deben coincidir usando la notación <alias>.<campo> (o <nombre tabla>.<campo>)

1.0.3 Right (outer) join

2. De igual manera, aunque quizás más contraintuitivo, podemos pegar a cada departamento los datos de los empleados que tengan igual id_departamento y NaN si no hay empleados en ese departamento. Quedaría así:

id_empleado	nombre	id_departamento	nombre_departamento
1	Ana	10	Marketing
2	Carlos	20	Ventas
3	Diana	10	Marketing
4	Eduardo	30	IT
NaN	NaN	40	Recursos Humanos

Antes aparecían todos los elementos de la izquierda y no necesariamente todos los elementos de la derecha y ahora es al contrario aparecen todos los elementos/filas de la tabla de la Çderecha y no todos los de la izquierda. Las tablas en ambos casos se completan con NaN/Nulls para los valores faltantes.

Es decir, aquí manda la tabla de la derecha. Es por eso que se denomina Right (outter) join.

Su sintáxis.

```
SELECT A.id_empleado, A.nombre, A.id_departamento, B.nombre_departamento
FROM Empleados AS A
RIGHT JOIN Departamentos AS B
ON A.id_departamento = B.id_departamento
```

Y las secciones y alias tienen la misma explicación que en el caso anterior.

1.0.4 Inner Join(cuando queremos una interseccion)

3. Un tercer caso, quizá más intuitivo que los dos anteriores, es quedarme solo con los datos de una y otra tabla que tienen id en las dos. Es decir la intersección por id_departamento en este caso, y quedaría algo así como:

id_empleado	nombre	id_departamento	nombre_departamento
1	Ana	10	Marketing
2	Carlos	20	Ventas
3	Diana	10	Marketing
4	Eduardo	30	IT

Fíjate que Luis que no tiene departamento o su id_departamento no aparece en Departamentos y Recursos Humanos el departamento que no aparece en los id_departamento de los empleados, se quedan fuera de la tabla resultante.

A esta ntersección se le llama INNER JOIN y su sintaxis es

```
SELECT A.id_empleado,A.nombre,A.id_departamento, B.nombre_departamento
FROM Empleados AS A
INNER JOIN Departamentos AS B
ON A.id_departamento = B.id_departamento
```

1.0.5 Full Outer Join

4. Igual que consideramos sólo las coincidencias podemos considerar unas y otras y hacer la unión de las salidas del left join y del right join:

id_empleado	nombre	id_departamento	nombre_departamento
1	Ana	10	Marketing
2	Carlos	20	Ventas
3	Diana	10	Marketing
4	Eduardo	30	IT
5	Luis	-1	NaN/Null
NaN/Null	Nan/Null	40	Recursos Humanos

En esta tabla ya aparecen los empleados sin departamento y los departamentos sin empleados eso sí con sus correspondientes campos a NaN

La sintaxis es similar a las anteriores pero ahora usan `FULL OUTTER JOIN`:

```
SELECT A.id_empleado,A.nombre,A.id_departamento, B.nombre_departamento
FROM Empleados AS A
FULL OUTER JOIN Departamentos AS B
ON A.id_departamento = B.id_departamento
```

1.0.6 Otros Joins

Existen otros joins como el producto cartesiano y los joins que se combinan con cláusulas where pero no los vamos a ver aquí y aqui tienes un par de tutoriales donde se amplia lo que hemos visto.

1.0.7 Lo importante

Lo importante es lo que uno quiera hacer con las tablas: * Que tengo una tabla que ”manda” y quiero completarla en lo posible y con nulos si no hay datos en las otras tablas de las que quiero

sacar información: Necesitas un LEFT o un RIGHT JOIN dependiendo del orden en el que pongas la tabla ”directora” (la que quieres completar), si la pones en el FROM, entonces LEFT JOIN si no, RIGHT JOIN * Que quieres quedarte solo con las filas que no vayan a tener nulos: INNER JOIN (sólo te quedaras con las filas o información de una y otra tabla que coincidan, las que producen nulos se quitan) * Que quieres mezclar las dos tablas al completo independientemente de donde se generen Nulos: FULL JOIN.

[ ]:

[ ]:

09_Joins_Ejemplos_II

November 29, 2023

1 SQL con Python: JOIN Ejemplos (II)

1.0.1 Preparación

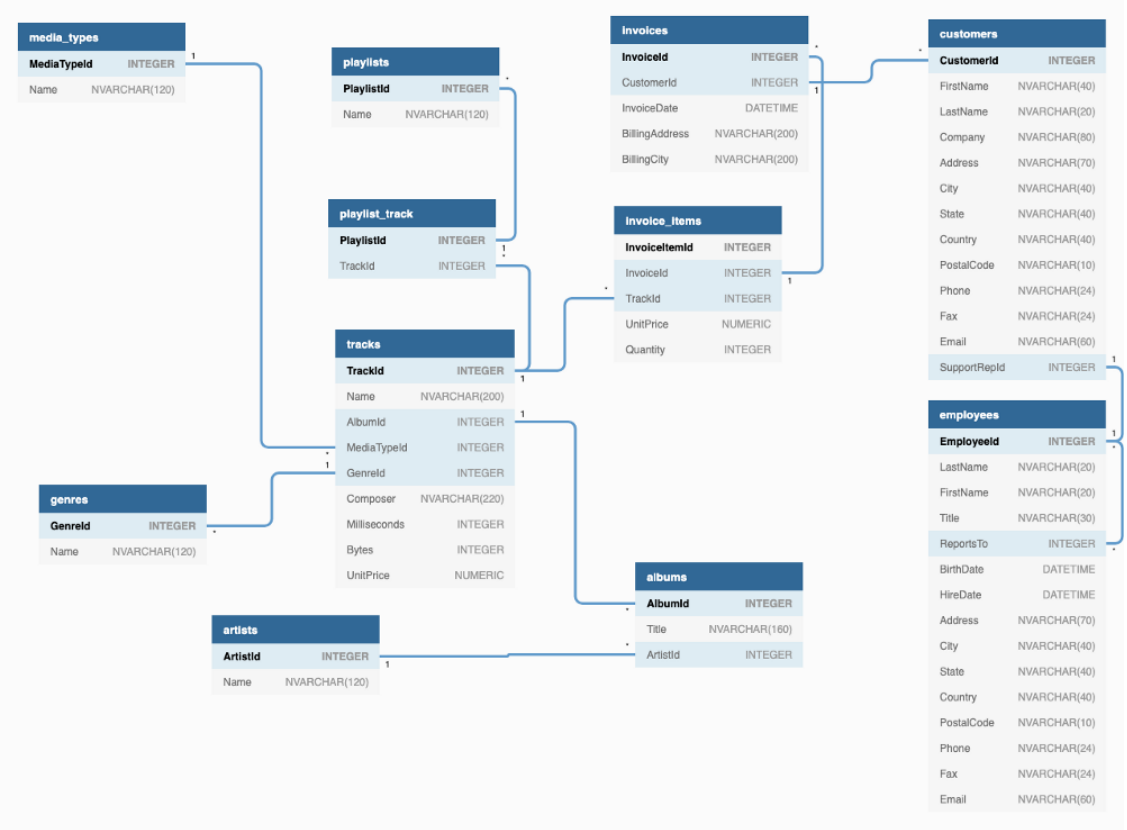
Como en la sesión anterior, importamos librerías, creamos la conexión a la base de datos, creamos el cursor y nuestra función para lanzar queries. Además también recuperamos el modelo de datos para tener claro que cruces pueden hacerse.

[1]:
```
import pandas as pd
import sqlite3

# Conectamos con la base de datos chinook.db
connection = sqlite3.connect("./data/chinook_joins.db")

# Obtenemos un cursor que utilizaremos para hacer las queries
cursor_bootcamp = connection.cursor()
```

[2]:
```
def sql_query(query):
    cursor_bootcamp.execute(query)
    ans = cursor_bootcamp.fetchall()
    names = [description[0] for description in cursor_bootcamp.description]
    return pd.DataFrame(ans,columns=names)
```



1.0.2 Right Join

Este es el JOIN en el que manda la tabla de la "derecha", la que ponemos detrás del `JOIN`. Es la tabla de la que queremos mantener todas las filas y pegarle campos de otra tabla aunque no haya cruce, en cuyo caso se rellenarán con nulos (NULL o NaN si lo pasamos a Pandas)

[ ]:
```
# Hagamos el inverso del anterior pero por la derecha: Queremos pegar a cada␣
 ↪disco las canciones
# Manda la tabla de álbumes (albums) (TB SE PUEDE PONER A LA DEREHCA)
# sobre el de canciones (tracks)
# Como en la sesión anterior la clave de cruce es AlbumId (Foreing_Key en␣
 ↪tracks, Primary_Key en albums)
```

[3]:
```
query ='''
SELECT b.*, a.name FROM tracks AS a RIGHT JOIN Albums AS b
on a.albumid = b.albumid
'''
df_rj = pd.read_sql(query, connection)
df_rj
```

[3]:

	AlbumId	Title	ArtistId	\
0	1	For Those About To Rock We Salute You	1	
1	2	Balls to the Wall	2	

```
2           3                                   Restless and Wild         2
3           3                                   Restless and Wild         2
4           3                                   Restless and Wild         2
…       …                                                   …       …
3498      343                              Respighi:Pines of Rome       226
3499      344  Schubert: The Late String Quartets & String Qu…       272
3500      345                                 Monteverdi: L'Orfeo       273
3501      346                               Mozart: Chamber Music       274
3502      347  Koyaanisqatsi (Soundtrack from the Motion Pict…       275

                                                  Name
0                For For Those About To Rock (We Salute You)
1                                      Balls to the Wall
2                                        Fast As a Shark
3                                      Restless and Wild
4                                   Princess of the Dawn
…                                                     …
3498  Pini Di Roma (Pinien Von Rom) \ I Pini Della V…
3499  String Quartet No. 12 in C Minor, D. 703 "Quar…
3500                L'orfeo, Act 3, Sinfonia (Orchestra)
3501  Quintet for Horn, Violin, 2 Violas, and Cello …
3502                                   Koyaanisqatsi

[3503 rows x 4 columns]
```

[4]:
```
df_rj.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3503 entries, 0 to 3502
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   AlbumId   3503 non-null   int64
 1   Title     3503 non-null   object
 2   ArtistId  3503 non-null   int64
 3   Name      3503 non-null   object
dtypes: int64(2), object(2)
memory usage: 109.6+ KB
```

Aquí las puntualizaciones son dos: 1. Aquí no hay albumid que sí esté en albums y no esté en tracks por eso no hay nulos en el campo **name** 2. Como ahora comprobaremos el número de regitros ha crecido porque ahora hay tantas lineas por album como canciones tenga el album, mientras que la tabla anterior de albums el campo AlbumId era único.

[6]:
```
query = '''
SELECT *
FROM Albums'''
```

```
df_albums =pd.read_sql(query, connection)
df_albums
```

[6]:

	AlbumId	Title	ArtistId
0	1	For Those About To Rock We Salute You	1
1	2	Balls to the Wall	2
2	3	Restless and Wild	2
3	4	Let There Be Rock	1
4	5	Big Ones	3
..	…	…	…
342	343	Respighi:Pines of Rome	226
343	344	Schubert: The Late String Quartets & String Qu…	272
344	345	Monteverdi: L'Orfeo	273
345	346	Mozart: Chamber Music	274
346	347	Koyaanisqatsi (Soundtrack from the Motion Pict…	275

[347 rows x 3 columns]

[7]:
```
df_albums.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 347 entries, 0 to 346
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   AlbumId   347 non-null    int64
 1   Title     347 non-null    object
 2   ArtistId  347 non-null    int64
dtypes: int64(2), object(1)
memory usage: 8.3+ KB
```

[9]:
```
# como eran los indices
print(df_albums.AlbumId.is_unique)
print(df_rj.AlbumId.is_unique)
# en el primero albumId es unico , solo  contando los albums( true)
# en el segundo ya no es unico pq se ha ampliado, hay un albun por cada cancion␣
  ↪que hay en ese album(false)
```

```
True
False
```

1.0.3 Inner Join(FILAS COMUNES A AMBAS TABLAS)

Veamos ahora un ejemplo de Inner Join, es decir aquel en el que sólo quiero quedarme con las filas que tengan valores comunes en las dos tablas (aquellas filas de una tabla cuyo identificador no aparezca en la clave de la otra no se tendrán en cuenta)

[13]:
```
# Vamos a quedarnos ahora con las canciones para las cuales si haya disco
# AHORA NOS vamos auedar con las canciones que tangan albun y los albun que␣
 ↪tenga canciones

query ='''
SELECT tracks.*, albums.title
FROM tracks
INNER JOIN albums -- aqui es indeferente que la info evnga de la derecha o la␣
 ↪ziq pq solo buscamos valores comunes a ambas
ON tracks.AlbumId =albums.AlbumId''' # OJO cuidado que aqui las columas se␣
 ↪llamen iguual , pero em michas ocasiones las claves se llamaran diferentes.␣
 ↪ejem: si para a esta columma se hubiera llamado lpId habria que ahbero␣
 ↪puesto para a

df_inner = pd.read_sql(query, connection)
df_inner.info() # no hay mas nulos sin contar el composer qu eya lo tenia desde␣
 ↪el principio
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3503 entries, 0 to 3502
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   TrackId       3503 non-null   int64
 1   Name          3503 non-null   object
 2   AlbumId       3503 non-null   int64
 3   MediaTypeId   3503 non-null   int64
 4   GenreId       3503 non-null   int64
 5   Composer      2525 non-null   object
 6   Milliseconds  3503 non-null   int64
 7   Bytes         3503 non-null   int64
 8   UnitPrice     3503 non-null   float64
 9   Title         3503 non-null   object
dtypes: float64(1), int64(6), object(3)
memory usage: 273.8+ KB
```

Aquí lo que podemos observar es: 1. No hay nulos, claro se cojen filas que existen en una y otra tabla. 2. Se ha reducido el número de salidas respecto al left join que hicimos (aunque no te des cuenta ahora si te haces una select de tracks podrás comprobar que tiene 3503 cacniones). Ha quitado las canciones que no tienen album.

1.0.4 Full join/ se queda todos los cruces posibles hasta los que haya nulos

El full join es el que se queda todos los cruces posibles (por valores existentes) aunque haya nulos. En el caso de sqlite3 no admite Full Join por lo que hay que simularlo. Como te dejo a continuación, pero dado que es un caso excepcional (trabajar con LEFT JOIN es casi lo más que vamos a hacer), no vamos a entrar en mucho más detalle

[Nota: Excepcional en nuestro Bootcamp/Data Science, puede que como usuario de SQL en tu empresa o en tu vida tengas que hacer miles de Full Outer Join]

[14]:
```
#FULL JOIN
#FULL JOIN isn't supported in sqlite,
#so we use a LEFT JOIN + RIGHT JOIN(inverse LEFT JOIN) as a workaround

query = '''
SELECT *
FROM invoice_items AS a
LEFT JOIN tracks AS b
ON a.trackid = b.trackid
UNION ALL -- usa el union all que permite unir  la salida de dos queries por␣
 ↪separado, no es una subquery una dentro de otra.( COMO HACER UN CONCAT EN␣
 ↪PANDAS SUMAS DF)
SELECT *
FROM tracks AS a
LEFT JOIN invoice_items AS b
ON a.trackid =b.trackid;
'''

sql_query(query)
```

[14]:

	InvoiceLineId		InvoiceId	\
0	1		1	
1	2		1	
2	3		2	
3	4		2	
4	5		2	
…	…		…	
5994	3500	String Quartet No. 12 in C Minor, D. 703 "Quar…		
5995	3500	String Quartet No. 12 in C Minor, D. 703 "Quar…		
5996	3501	L'orfeo, Act 3, Sinfonia (Orchestra)		
5997	3502	Quintet for Horn, Violin, 2 Violas, and Cello …		
5998	3503	Koyaanisqatsi		

	TrackId	UnitPrice	Quantity	TrackId	\
0	2	0.99	1	2	
1	4	0.99	1	4	
2	6	0.99	1	6	
3	8	0.99	1	8	
4	10	0.99	1	10	
…	…	…	…	…	
5994	344	2.00	24	Franz Schubert	
5995	344	2.00	24	Franz Schubert	
5996	345	2.00	24	Claudio Monteverdi	
5997	346	2.00	24	Wolfgang Amadeus Mozart	

```
5998     347       2.00        10              Philip Glass

                      Name  AlbumId  MediaTypeId  GenreId  \
0          Balls to the Wall        2         2.00      1.0
1          Restless and Wild        3         2.00      1.0
2     Put The Finger On You         1         1.00      1.0
3           Inject The Venom        1         1.00      1.0
4                 Evil Walks        1         1.00      1.0
…                        …      …           …       …
5994                 139200  2283131         0.99    578.0
5995                 139200  2283131         0.99   1727.0
5996                  66639  1189062         0.99      NaN
5997                 221331  3665114         0.99      NaN
5998                 206005  3305164         0.99      NaN

                                                Composer  Milliseconds  \
0                                                   None      342562.0
1     F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D…      252051.0
2              Angus Young, Malcolm Young, Brian Johnson      205662.0
3              Angus Young, Malcolm Young, Brian Johnson      210834.0
4              Angus Young, Malcolm Young, Brian Johnson      263497.0
…                                                    …           …
5994                                               108        3500.0
5995                                               319        3500.0
5996                                              None           NaN
5997                                              None           NaN
5998                                              None           NaN

          Bytes  UnitPrice
0     5510424.00       0.99
1     4331779.00       0.99
2     6713451.00       0.99
3     6852860.00       0.99
4     8611245.00       0.99
…          …        …
5994        0.99       1.00
5995        0.99       1.00
5996         NaN        NaN
5997         NaN        NaN
5998         NaN        NaN

[5999 rows x 14 columns]
```

[ ]:

10_Gestion_BD

November 29, 2023

1 SQL en Python: Gestion BD (I)

La gestión de Bases de Datos es un mundo muy amplio que no podemos abarcar en un sprint y mucho menos en un par de píldoras, pero sí es bueno por lo menos que conozcas cómo con SQL (y para el gestor Sqlite3) se pueden hacer acciones tales como:
* Insertes, Updates y Deletes en las tablas * Crear y Tirar Tablas * Usar vistas

Que son los puntos que vamos a tratar aunque no necesariamente en ese orden, de hecho empezaremos creando nuestra base de datos y luego las tablas dentro de ellas

1.0.1 Creación de Bases de Datos y tablas

Podemos crear nuestras propias bases de datos utilizando SQLite (que no es directamente trasladable a otros gestores) y luego dentro crearemos tablas usando ya sí sintaxis SQL. Nos importamos nuestras librerías y comenzamos a trabajar:

[1]:
```
import pandas as pd
import sqlite3
```

Ahora haremos una conexión a una base de datos que en realidad no existe, pero esa es la forma de crearla:

[2]:
```
connection = sqlite3.connect("Base de Datos _I.db")
```

Y ya podemos crearnos el cursor para poder interactuar con ella:

[3]:
```
cursor_gestion = connection.cursor()
```

Y ahora ya podemos crear una tabla siguiendo la siguiente sintaxis:

```
CREATE TABLE nombre_tabla (
    columna1 tipo_de_dato restricciones,
    columna2 tipo_de_dato restricciones,
    ...
    columnaN tipo_de_dato restricciones
);
```

Donde:

- **nombre_tabla:** Es el nombre de la tabla que queremos crear.
- **columna1, columna2, ..., columnaN:** Son los nombres de las columnas de la tabla.
- **tipo_de_dato:** Es el tipo de dato de cada columna (por ejemplo, `INT` para enteros, `VARCHAR` o `CHAR` para cadenas de texto, `DATE` para fechas, etc.).
- **restricciones:** Son las restricciones o reglas para cada columna (opcional). Algunas restricciones comunes incluyen `PRIMARY KEY`, `NOT NULL`, `UNIQUE`, `FOREIGN KEY`, etc. (que corresponden a un índice único, a que no puede dejarse vacío el campo, a que no puede repetirse o a que tendrá que enlazarse con la clave en otra tabla)

Creemos una tabla sencilla de alumnos de un Master: (te lo doy hecho)

[4]:
```
query_create = '''
CREATE TABLE Master_Class (
ID       INT PRIMARY KEY, -- nombre tipo_de_dato restricción como hemos visto␣
 ↪antes
NOMBRE   TEXT NOT NULL,
EDAD     INT NOT NULL,
CIUDAD   CHARS(50), -- Le decimos que este campo siempre tiene 50 caracteres, y␣
 ↪los que no llene los rellenará con espacios
NOTAS    FLOAT
)
'''
cursor_gestion.execute(query_create)
```

[4]:
```
<sqlite3.Cursor at 0x228864745c0>
```

Y una vez la tenemos, podemos buscarla en la tabla maestra:

[6]:
```
query = "SELECT * FROM sqlite_master WHERE type = 'table'"
cursor_gestion.execute(query)
cursor_gestion.fetchall()
```

[6]:
```
[('table',
  'Master_Class',
  'Master_Class',
  2,
  'CREATE TABLE Master_Class (\nID       INT PRIMARY KEY, -- nombre tipo_de_dato
restricción como hemos visto antes\nNOMBRE   TEXT NOT NULL,\nEDAD     INT NOT
NULL,\nCIUDAD   CHARS(50), -- Le decimos que este campo siempre tiene 50
caracteres, y los que no llene los rellenará con espacios\nNOTAS    FLOAT\n)')]
```

Fíjate que almacena el nombre, y la query empleada para crearla. EN las tablas maestras demas de los campos se guarda la quuery que ha creado ea BD

1.0.2 Insert

Ahora que tenemos una tabla podemos ingestar registros (nombre técnico), modificarlos y borrarlos tal como vimos de forma teórica hace ya unas cuantas sesiones. En esta píldora trataremos el insert y dejaremos update y delete para la próxima.

Recordemos la sintaxis de un insert:

```
INSERT INTO nombre_tabla (columna1, columna2, columna3, ...)
VALUES (valor1, valor2, valor3, ...);
```

Donde:

- **nombre_tabla:** Nombre de la tabla donde se insertarán los datos.
- **columna1, columna2, ... :** Las columnas de la tabla en las que se insertarán los datos. No es necesario incluir todas las columnas, especialmente si algunas tienen valores predeterminados o son autoincrementables.
- **valor1, valor2, ... :** Los valores correspondientes a las columnas especificadas. Deben estar en el mismo orden que las columnas y deben ser del tipo de dato adecuado para cada columna.

Insertemos unos cuantos valores:

- Luis, 24, Madrid, 8.5
- Ana, 32, Lugo, 6.25
- Juan, 35, Bilbao, 5.55
- Nuria, 41, Alicante, 9.75

[14]:
```
# Usa este diccionario para no ir valor a valor:
datos = {
    "Luis": (24,"Madrid", 8.5),
    "Ana": (32,"Lugo", 6.25),
    "Juan": (35, "Bilbao", 5.55),
    "Nuria": (51, "Alicante", 9.75)
}
for indice, (nombre, valores) in enumerate (datos.items()):# la primera vez␣
 ↪quie hacemos un enumerate de un items. el items ya nos da clave valor , pero␣
 ↪si le pasamos al enumerate el items, nos dara el indice  mas una tupla con 2␣
 ↪valores, y si quiero capturarla la tengo que poner ais
    edad = valores [0]
    ciudad = valores[1]
    nota= valores[2]
    query = f"INSERT INTO Master_Class (ID, NOMBRE, EDAD, CIUDAD, NOTAS) VALUES␣
 ↪({indice}, '{nombre}', {edad}, '{ciudad}', {nota})"

    cursor_gestion.execute(query)
```

Y, esto es importante, ahora es necesario hacer algo que no habíamos hecho hasta ahora, y que se parece a lo que hacemos con los repos, hay que confirmarle a nuestro gestor que queremos hacer

los cambios:

[15]:
```
connection.commit()
```

Para comprobar que realmente se han hecho los cambios haríamos nuestra consulta y podríamos pasarlo a un `DataFrame` pero vaya, no tenemos nuestra función, no importa porque ya es hora de que te enseñe otra cosa:

[16]:
```
query = '''
SELECT * FROM Master_Class '''

df = pd.read_sql(query, connection)

df
```

[16]:

	ID	NOMBRE	EDAD	CIUDAD	NOTAS
0	0	Luis	24	Madrid	8.50
1	1	Ana	32	Lugo	6.25
2	2	Juan	35	Bilbao	5.55
3	3	Nuria	51	Alicante	9.75

[17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ID      4 non-null      int64
 1   NOMBRE  4 non-null      object
 2   EDAD    4 non-null      int64
 3   CIUDAD  4 non-null      object
 4   NOTAS   4 non-null      float64
dtypes: float64(1), int64(2), object(2)
memory usage: 288.0+ bytes
```

[18]:
```
connection.close()# siempre hay que hacer esto para evitar usar recursos␣
 ↪innecesarios
```

11_Gestion_BDs_II

November 29, 2023

1 SQL en Python: Gestion BD (II)

En esta sesión completamos nuestra pequeña inmersión en el mundo del SQL y las bases de datos relacionales. Lo primero es volver a crearnos nuestra base de datos de la sesión anterior. Aquí te dejo las celdas preparadas:

[1]:
```
import pandas as pd
import sqlite3
```

[2]:
```
connection = sqlite3.connect("gestion_sesion.db")
```

[3]:
```
cursor_gestion = connection.cursor()
```

[4]:
```
query_create = '''
CREATE TABLE Master_Class (
ID       INT PRIMARY KEY, -- nombre tipo_de_dato restricción como hemos visto␣
 ↪antes
NOMBRE   TEXT NOT NULL,
EDAD     INT NOT NULL,
CIUDAD   CHARS(50), -- Le decimos que este campo siempre tiene 50 caracteres, y␣
 ↪los que no llene los rellenará con espacios
NOTAS    FLOAT
)
'''
cursor_gestion.execute(query_create)
```

[4]:
```
<sqlite3.Cursor at 0x28fa4a06040>
```

[5]:
```
# Usa este diccionario para no ir valor a valor:
datos = {
    "Luis": (24,"Madrid", 8.5),
    "Ana": (32,"Lugo", 6.25),
    "Juan": (35, "Bilbao", 5.55),
    "Nuria": (51, "Alicante", 9.75)
}
for indice,(nombre,valores) in enumerate(datos.items()):
    edad = valores[0]
    ciudad = valores[1]
    notas = valores[2]
    query = f"INSERT INTO Master_Class (ID,NOMBRE,EDAD,CIUDAD,NOTAS) VALUES␣
 ↪({indice},'{nombre}',{edad},'{ciudad}',{notas})"
    cursor_gestion.execute(query)
```

[6]:
```
connection.commit()# HAce falta haer siempre commit los insert, update o delte␣
 ↪no
```

1.0.1 Update

La sintaxis general de UPDATE es:

```
UPDATE nombre_tabla
SET columna1 = valor1, columna2 = valor2, ...
WHERE condicion;
```

- **nombre_tabla:** Es el nombre de la tabla donde se realizarán las modificaciones.
- **SET columna1 = valor1, columna2 = valor2, ... :** Aquí se listan las columnas que se van a actualizar y los nuevos valores que se les asignarán. Puedes actualizar una o varias columnas a la vez.
- **WHERE condicion:** Especifica qué registros deben ser actualizados. La condición puede ser cualquier expresión lógica válida. Si omites la cláusula WHERE, todos los registros en la tabla serán actualizados, lo cual puede tener consecuencias no deseadas.

Modifiquemos en nuestra tabla la edad de Nuria por 48 y comprobemos el resultado utlizando `read_sql`

[8]:
```
query ='''
UPDATE Master_Class
SET edad = 48
WHERE nombre = "Nuria"'''

#hacemo la ejecucion del cambio con el cursor
cursor_gestion.execute(query)
```

```
# la comprobacion del cambio en el df con toda la info pero no aplica los␣
 ↪cambios hasta que no hagamos commmit
pd.read_sql("SELECT* FROM Master_Class", connection)
```

[8]:

	ID	NOMBRE	EDAD	CIUDAD	NOTAS
0	0	Luis	24	Madrid	8.50
1	1	Ana	32	Lugo	6.25
2	2	Juan	35	Bilbao	5.55
3	3	Nuria	48	Alicante	9.75

Pero ojo no nos dejemos engañar, necesitamos hacer el commit igualmente para que el cambio se grabe en la base de datos:

[9]:
```
connection.commit()
```

Si hubieramos intentado actualizar alguna columna incumpliendo las restricciones con las que fueron creadas hubiera dado error.

1.0.2 Delete

DELETE nos permite borrar filas de las tablas, y sigue esta sintaxis:

```
DELETE FROM nombre_tabla WHERE condicion;
```

- **nombre_tabla:** Es el nombre de la tabla de la cual se eliminarán los registros.
- **condicion:** Especifica qué registros deben ser eliminados. Puede ser cualquier expresión lógica válida.

Si omites la cláusula WHERE en una instrucción DELETE, todos los registros de la tabla especificada serán eliminados, lo cual debe hacerse con precaución.

Borremos las filas de los alumnos con menos de un 7:

[10]:
```
query = '''DELETE FROM Master_Class '''# si hacemos commit nos lo cargamos todo

#hacemo la ejecucion del cambio con el cursor
cursor_gestion.execute(query)
```

[10]:
```
<sqlite3.Cursor at 0x28fa4a06040>
```

[11]:
```
# la comprobacion del cambio en el df con toda la info pero no aplica los␣
 ↪cambios hasta que no hagamos commmit
pd.read_sql("SELECT* FROM Master_Class", connection)
```

[11]:
```
Empty DataFrame
Columns: [ID, NOMBRE, EDAD, CIUDAD, NOTAS]
Index: []
```

Eh, que susto, tranquilidad, como con `INSERT` y `UPDATE` es necesario hacer el commit para que los cambios se hagan efectivos. En este caso cierra la conexión a la base de datos y abre una nueva.

[14]:
```
#como no quiero borrarla, para recuperarla tengo que cerrar conxexion con la BD
 ↪y vover a conectarme de nuevo haciendo: close, conectar, abrir de nuevo el
 ↪cursos
connection.close()
```

[15]:
```
#connection = sqlite3.connect("gestion_sesion.db")
connection = sqlite3.connect("gestion_sesion.db")
# conectamos cursor
cursor_gestion = connection.cursor()
```

[16]:
```
pd.read_sql("SELECT* FROM Master_Class", connection)# no llegamos a borrar pq
 ↪no hicimos el commit
```

[16]:

	ID	NOMBRE	EDAD	CIUDAD	NOTAS
0	0	Luis	24	Madrid	8.50
1	1	Ana	32	Lugo	6.25
2	2	Juan	35	Bilbao	5.55
3	3	Nuria	48	Alicante	9.75

Y ahora sí borremos esas filas: nota < 7

[17]:
```
query = '''DELETE FROM  Master_Class WHERE notas < 7'''
# ejecutams cursor
cursor_gestion.execute(query)
# mostramos los cambios como quedarian
pd.read_sql("SELECT * FROM Master_Class", connection)
```

[17]:

	ID	NOMBRE	EDAD	CIUDAD	NOTAS
0	0	Luis	24	Madrid	8.50
1	3	Nuria	48	Alicante	9.75

De nuevo, hay que hacer el commit, hagámoslo:

[18]:
```
connection.commit()
```

[ ]:
```

```

1.0.3 Borrado de columnas

Igual que en pandas tenemos el método `drop` con su argumento columns para borrar columnas en SQL tenemos:

```
ALTER TABLE nombre_tabla DROP COLUMN nombre_columna;
```

- **nombre_tabla:** Es el nombre de la tabla de la que deseas eliminar una columna.
- **nombre_columna:** Es el nombre de la columna que deseas eliminar.

Nota Importante: - Algunos sistemas de bases de datos pueden requerir modificadores adicionales o tener restricciones específicas para eliminar columnas. Por ejemplo, si otras tablas dependen de la columna que estás intentando eliminar (a través de claves foráneas, por ejemplo), es posible que necesites modificar o eliminar esas dependencias antes de poder eliminar la columna. - Siempre

es una buena práctica hacer una copia de seguridad de tu base de datos antes de realizar cambios estructurales como eliminar columnas.

Probemos a borrar la columna "edad":

[24]:
```
query = ''' ALTER TABLE Master_Class DROP COLUMN edad'''

cursor_gestion.execute(query)
```

[24]:
```
<sqlite3.Cursor at 0x28fa4eeb7c0>
```

[25]:
```
pd.read_sql("SELECT * FROM MAster_Class", connection)
```

[25]:

	ID	NOMBRE	CIUDAD	NOTAS
0	0	Luis	Madrid	8.50
1	3	Nuria	Alicante	9.75

Pero recuerdad que si no haces el commit tampoco se grabará el cambio, ejecuta lo siguiente:

[26]:
```
connection.commit
```

[26]:
```
<function Connection.commit()>
```

Tienes que hacer el commit después del ALTER/DROP.

1.0.4 Borrado de tablas

Para terminar la sesión y el grupo dedicado a SQL y bases de datos, veamos como borrar una tabla:

```
DROP TABLE nombre_tabla;
```

Algunas consideraciones importantes:

1. **Cuidado con los Datos:** Al usar `DROP TABLE`, la tabla y todos sus datos se eliminan permanentemente. No hay forma de deshacer esta acción en la mayoría de los sistemas de gestión de bases de datos. Asegúrate de que realmente deseas eliminar la tabla y de que has realizado una copia de seguridad de los datos si es necesario.
2. **Dependencias:** Si la tabla está referenciada por otras tablas a través de claves foráneas u otros mecanismos de integridad referencial, es posible que primero debas eliminar o modificar estas dependencias.
3. **Sintaxis Específica del SGBD:** La sintaxis básica de `DROP TABLE` es bastante uniforme en los diferentes sistemas de gestión de bases de datos (SGBD), pero algunos pueden ofrecer opciones adicionales. Por ejemplo, en algunos SGBD puedes usar `DROP TABLE IF EXISTS nombre_tabla;` para evitar errores si la tabla no existe.
4. **Permisos:** Necesitas tener los permisos adecuados en la base de datos para eliminar tablas. Si no tienes los permisos necesarios, la operación fallará.

[27]:
```
query = '''DROP TABLE Master_Class'''

cursor_gestion.execute(query)
```

[27]:
```
<sqlite3.Cursor at 0x28fa4eeb7c0>
```

Veamos que pasa con la master table de Sqlite donde debería haber desaparecido:

[28]:
```
pd.read_sql("SELECT Name FROM sqlite_master WHERE type = 'table'", connection)
```

[28]:
```
Empty DataFrame
Columns: [name]
Index: []
```

[29]:
```
connection.commit()
```

De nuevo tendrás que hacer commit para asegurarte de que se borra de verdad. Ojo, en algunos gestores al cerrar el gestor o una base de datos se hace un commit automático así que no te fíes después de haber borrado accidentalmente cierra la base de datos pero evitando que se hagan commits.