# 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 útil para ejecutar queries:

[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()

[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*

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)

### 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)

[3]:

*# Obtengamos los datos de la tabla "tracks" ordenados por nombre de forma descendente*

query**=**'''

SELECT \*

FROM tracks

ORDER BY name DESC

'''

sql\_query(query)

[3]:

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

# 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

'''

sql\_query(query)

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente con confianza media

### 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](https://www.sqlservertutorial.net/sql-server-aggregate-functions/) encontrarás un resumen con las principales funciones.

[5]:

*# Obtener el número de canciones que contienen la palabra love*

query**=**'''

SELECT COUNT(\*)

FROM tracks

WHERE name LIKE "%love%"

'''

sql\_query(query)

[5]:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

# Encontrar la media del precio de las canciones compradas

query='''

SELECT AVG(unitprice)

FROM invoice\_items

'''

sql\_query(query)

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

# Obtener la duración máxima de las canciones

query='''

SELECT MAX(milliseconds/1000)

FROM tracks

'''

sql\_query(query)

Imagen que contiene Carta

Descripción generada automáticamente

query='''

SELECT name

FROM tracks

WHERE milliseconds/1000=5286

'''

sql\_query(query)

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

### Agrupaciones (GROUP BY):

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

*# Selección del precio unitario en funcion del género*

query**=**'''

SELECT GenreID, SUM(unitprice) AS "Total\_Price"

FROM tracks

GROUP BY GenreId

ORDER BY Total\_Price DESC

LIMIT 10

'''

sql\_query(query)

Tabla

Descripción generada automáticamente con confianza media

O calcular los 10 compositores con más canciones y ordenarlas de mayor a menor

query**=**'''

SELECT Composer AS "Compositor", COUNT(\*) AS "Total Canciones"

FROM tracks

WHERE composer IS NOT NULL

GROUP BY Composer

ORDER BY 2 DESC

LIMIT 10

'''

sql\_query(query)

Tabla

Descripción generada automáticamente