Status	Finished
Started	Wednesday, 4 December 2024, 12:23 PM
Completed	Wednesday, 4 December 2024, 12:35 PM
Duration	11 mins 8 secs
Marks	16.00/16.00
Grade	**10.00** out of 10.00 (**100**%)

Information

¿Qué es SQL?

SQL es un lenguaje estándar para acceder y manipular bases de datos. SQL se convirtió en un estándar del American National Standards Institute (ANSI) en 1986 y de la Organización Internacional de Normalización (ISO) en 1987.

Las siglas SQL significan Lenguaje de Consulta Estructurado (**S**tructured **Q**uery **L**anguage).

Usando el lenguaje SQL se pueden crear nuevas bases de datos, crear tablas, ejecutar consultas, recuperar datos, insertar, actualizar y eliminar registros, etc.

Existen diferentes versiones del lenguaje SQL pero, para cumplir con el estándar, todos admiten al menos los comandos principales.

Python puede usarse para desarrollar programas que manipulen bases de datos. Estos programas se conectan con un Sistema de Gestión de Bases de Datos (DBMS, en inglés), usando un *conector*, e interactúan con las bases de datos por medio de *comandos* SQL.

En este tutorial vamos a usar SQLite 3 como Sistema de Gestión de Bases de Datos. Para interactuar con SQLite 3 desde un programa escrito en Python, debemos importar el módulo *sqlite3*:

```
import sqlite3
```

Information

Crear y eliminar BBDD

La manipulación de bases de datos usando el lenguaje SQL se realiza mediante sentencias que se componen de una o varias instrucciones.

Para crear una base de datos se usa el comando

CREATE DATABASE *nombre*;

Por ejemplo, la siguiente sentencia crea una base de datos llamada *FormulaUno*:

```
CREATE DATABASE FormulaUno;
```

Aunque, por convención, en este tutorial pondremos en mayúscula las instrucciones SQL, el lenguaje no hace distinción entre mayúsculas y minúsculas. Obsérvese que la sentencia acaba en punto y coma, esto es obligatorio en algunos SGBD y no en otros, pero es el estándar para **separar** las sentencias SQL cuando queremos indicar una secuencia de ellas.

Desde Python conectamos con una base de datos SQLite usando la función **`connect`**:

```
db = sqlite3.connect("FormulaUno")
```

Si la base de datos existe, la función `connect` abre una *conexión* con la misma y la devuelve, quedando en el ejemplo asignada a la variable db para su uso posterior. Si la base de datos no existe, la crea primero y luego abre la conexión. Si no puede crearla se produce un error.

Cuando terminemos de trabajar con una base de datos debemos cerrar la conexión correspondiente:

```
db.close()
```

También se puede establecer la conexión usando una cláusula de contexto (with), de esta manera, la propia cláusula de contexto se encargará de cerrar la conexión cuando termine su ámbito, evitando que tengamos que escribir la instrucción de cierre.

```
with sqlite3.connect("FormulaUno") as db:
    <Operaciones con la base de datos>
```

No existe un comando para eliminar una base de datos en *sqlite3*. Como la base de datos se almacena en un fichero con la **extensión `db`** (p.e. `FormulaUno.db`), basta con eliminar ese fichero. En el estándar SQL se usa el comando `DROP DATABASE nombre;`. Por ejemplo:

```
DROP DATABASE FormulaUno;
```

Question 1

Complete

Mark 2.00 out of 2.00

Complete la siguiente instrucción para establecer una conexión con una base de datos SQLite 3 llamada "FormulaUno":

```
db = [ sqlite3 ].[ connect ]("FormulaUno")
```

Information

Creación de tablas

Para crear una tabla en la base de datos activa, se usa el comando:

CREATE TABLE *table_name* (*columna1 tipo1, columna2 tipo2, columna3 tipo3, ...*)

Por ejemplo:

```
CREATE TABLE Pilotos (
    id INTEGER,
    Piloto TEXT(20),
    Escudería TEXT(20),
    País TEXT(20),
    Puntos INTEGER,
    Victorias INTEGER
)
```

Crea una tabla cuyos registros tienen la estructura:

id	Piloto	Escudería	País	Puntos	Victorias

En un entorno interactivo proporcionado por un SGBD, los comando SQL se escriben tal cual. Para usarlos desde Python, se debe crear un **cursor** asociado a la base de datos:

```
db = sqlite3.connect("FormulaUno")
cursor = db.cursor()
```

Una vez creado el cursor, se usa para ejecutar los comandos SQL, que se le pasan como parámetros de tipo string:

```
sql = """
    CREATE TABLE Pilotos (
        id INTEGER, Piloto TEXT(20),
        Escudería TEXT(20),
        País TEXT(20),
        Puntos INTEGER,
        Victorias INTEGER
    );
"""
cursor.execute(sql)
```

En la creación de la tabla se usan los tipos de datos INTEGER y TEXT. SQLite usa los tipos de datos: NULL, INTEGER, REAL, NUMERIC Y BLOB; otros SGBD pueden ofertar una variedad de tipos más amplia. La siguiente tabla muestra la equivalencia entre los tipos de Python y los de SQLite:

Python type	**SQLite type**
None	NULL
int	INTEGER
long	INTEGER
float	REAL
str (UTF8-encoded)	TEXT
unicode	TEXT
buffer	BLOB

Para **eliminar** una tabla se usa el comando

DROP TABLE *table_name*

Por ejemplo:

```
DROP TABLE Pilotos;
```

Question 2

Complete

Mark 3.00 out of 3.00

Complete el siguiente código para crear una tabla en una base de datos SQLite 3 desde un programa escrito en Python:

```
cursor = db.[ cursor ]()
```

```
sql = """
    CREATE [ TABLE ] Pilotos (
        id INTEGER, Piloto TEXT(20),
        Escudería TEXT(20),
        País TEXT(20),
        Puntos INTEGER,
        Victorias INTEGER
    );
"""
cursor.[ execute ](sql)
```

Information

La instrucción INSERT INTO

Para añadir registros en una tabla de la base de datos se usa la instrucción

INSERT INTO *Tabla* (*columna1*, *columna2*, ...) VALUES (*valor1*, *valor2*, ...)

Por ejemplo:

```
INSERT INTO Pilotos VALUES (1, 'Hamilton', 'Mercedes', 'Inglaterra', 250, 8)
```

En Python:

```
sql = "INSERT INTO Pilotos VALUES (?, ?, ?, ?, ?, ?)"
values = (1, 'Hamilton', 'Mercedes', 'Inglaterra', 250, 8)
cursor.execute(sql, values)
```

Nótese, que cuando se usa SQL desde un programa (en cualquier lenguaje, no sólo Python), **no se deben incluir los valores** de los campos en la string del comando SQL, sino que deben proporcionarse como una tupla separada, sustituyéndolos por interrogaciones en la sentencia SQL; esto es por razones de seguridad, ya que si no, se posibilitaría una intrusión no autorizada a la base de datos mediante "inyección de código".

Existe la opción de no proporcionar valores para todas las columnas, en cuyo caso hay que especificar en el comando INSERT INTO las columnas para las que se proporcionan valores:

```
sql = "INSERT INTO Pilotos (id, Piloto, Escudería, Puntos) VALUES (?, ?, ?, ?)"
values = (2, 'Bottas', 'Mercedes', 188)
cursor.execute(sql, values)
```

En el ejemplo anterior, se omiten las columnas *País* y *Victorias*, que quedan almacenadas en la base de datos con el valor *NULL*, equivalente al valor *None* de Python.

Question 3

Complete

Mark 2.00 out of 2.00

Complete el siguiente código para insertar un registro en una tabla de una base de datos SQLite 3 desde Python:

```
sql = "INSERT INTO Pilotos VALUES (?, ?, ?, ?, ?, ?)"
values = (1, 'Hamilton', 'Mercedes', 'Inglaterra', 250, 8)
cursor.execute(sql, values)
```

Information

La instrucción SELECT

Para recuperar información de una base de datos se usa la instrucción

SELECT *columna1*, *columna2*, ... FROM *Tabla*

Por ejemplo:

```
SELECT Piloto, Puntos FROM Pilotos
```

El resultado de una instrucción SELECT se almacena en una tabla temporal, el *result_set*, que, en un entorno interactivo, normalmente se muestra al usuario, o bien se usa como parte de otro comando. Para ejecutar la instrucción SELECT desde Python hay que recurrir al consabido cursor:

```
sql = "SELECT Piloto, Puntos FROM Pilotos"
cursor.execute(sql)
```

Para acceder luego, desde Python, al resultado de la sentencia SELECT, se debe ejecutar el método **`fetchall()`** del cursor, lo que proporciona una lista de tuplas, cada una de las cuales representa un registro del *result_set*:

```
result_set = cursor.fetchall()
for item in result_set:
    print(item)
```

Suponiendo la siguiente tabla *Pilotos*:

id	Piloto	Escudería	País	Puntos	Victorias
1	L. Hamilton	Mercedes	Inglaterra	250	8
2	V. Bottas	Mercedes	Finlandia	188	2
3	M Verstappen	Red Bull	Holanda	181	2
4	S. Vettel	Ferrari	Alemania	156	0
5	C. Leclerc	Ferrari	Mónaco	132	0
6	P. Gasly	Red Bull	Francia	63	0
7	C. Sainz Jr.	McLaren	España	58	0
8	K, Raikkonen	Alfa Romeo	Finlandia	31	0
9	D. Kvyat	Toro Rosso	Rusia	27	0
10	L. Norris	McLaren	Inglaterra	24	0

El resultado del ejemplo anterior sería:

```
('Hamilton', 250)
('Bottas', 188)
('Verstappen', 181)
('Vettel', 156)
('Leclerc', 132)
('Gasly', 63)
('Sainz Jr.', 58)
('Raikkonen', 31)
('Kvyat', 27)
('Norris', 24)
```

Nótese que cada registro del *result_set* es siempre una tupla, aunque tenga un solo elemento; si en el ejemplo anterior cambiamos la sentencia SELECT para obtener sólo los pilotos:

```
sql = "SELECT Piloto FROM Pilotos"
cursor.execute(sql)
result_set = cursor.fetchall()
```

Se obtiene el siguiente valor para *result_set*:

```
[('Hamilton',), ('Bottas',), ('Verstappen',), ('Vettel', ), ('Leclerc',), ('Gasly',), ('Sainz Jr.',), ('Raikkonen',), ('Kvyat',),
('Norris',)]
```

El primer piloto no es `result_set[0]`, sino `result_set[0][0]`

Si se quieren recuperar todas las columnas, basta con poner un **asterisco** en vez de enumerarlas:

```
SELECT * FROM Pilotos;
```

A veces, el resultado de una instrucción SELECT puede tener registros repetidos; por ejemplo:

```
SELECT Escudería FROM Pilotos;
```

da como resultado :

```
[('Mercedes',), ('Mercedes',), ('Red Bull',), ('Red Bull',), ('McLaren',), ('Alfa Romeo',), ('Toro Rosso',), ('McLaren',)]
```

Si queremos evitar las repeticiones, podemos añadir a la instrucción SELECT la cláusula **DISTINCT**:

```
SELECT DISTINCT Escudería FROM Pilotos;
```

y obtenemos:

```
[('Mercedes',), ('Red Bull',), ('McLaren',), ('Alfa Romeo',), ('Toro Rosso',)]
```

Question 4

Complete

Mark 2.00 out of 2.00

Complete el siguiente código para recuperar los pilotos y las escuderías de la tabla Pilotos de una base de datos SQLite 3 desde Python:

```
sql = " SELECT  Piloto, Escudería  FROM  Pilotos"
cursor.execute(sql)
```

Information

La cláusula WHERE

La cláusula WHERE se usa para establecer condiciones para filtrar los datos de una búsqueda. Por ejemplo, dada la siguiente tabla *Pilotos*:

id	Piloto	Escudería	País	Puntos	Victorias
1	Hamilton	Mercedes	Inglaterra	250	8
2	Bottas	Mercedes	Finlandia	188	2
3	Verstappen	Red Bull	Holanda	181	2
4	Vettel	Ferrari	Alemania	156	0
5	Leclerc	Ferrari	Mónaco	132	0
6	Gasly	Red Bull	Francia	63	0
7	Sainz Jr.	McLaren	España	58	0
8	Raikkonen	Alfa Romeo	Finlandia	31	0
9	Kvyat	Toro Rosso	Rusia	27	0
10	Norris	McLaren	Inglaterra	24	0

El código:

```
sql = "SELECT Piloto, Puntos FROM Pilotos WHERE Puntos > 150"
cursor.execute(sql)
select_result = cursor.fetchall()
for item in select_result:
    print(item)
```

Da como resultado:

```
('Hamilton', 250)
('Bottas', 188)
('Verstappen', 181)
('Vettel', 156)
```

La condición de la cláusula WHERE puede ser compuesta:

```
SELECT Piloto, Puntos FROM Pilotos WHERE Puntos > 150 AND Victorias > 2
```

En las condiciones de una cláusula WHERE se pueden usar:

- Los operadores booleanos (AND, OR, NOT)
- Los operadores relacionales (<, <=, =, <>, !=, >= >)
- Otros operadores (BETWEEN, LIKE, IN)

(los operadores de desigualdad <> y != son equivalentes)

El operador BETWEEN sirve para establecer un rango:

```
SELECT Piloto, Puntos FROM Pilotos WHERE Puntos BETWEEN 150 AND 200
```

El operador LIKE sirve para establecer un patrón con comodines. En SQLite se pueden usar dos comodines, el carácter '%', que significa "cero o más caracteres" y el carácter '_', que significa "un carácter". Por ejemplo:

```
SELECT Piloto, Puntos FROM Pilotos WHERE Piloto LIKE '%n'
```

encuentra todos los registros cuyos pilotos tienen un nombre termina con la letra 'n', precedida de cualquier número de caracteres (Hamilton, Verstappen y Raikkonen).

Mientras que:

```
SELECT Piloto, Puntos FROM Pilotos WHERE País LIKE '%l_n%'
```

Encuentra los registros cuyo campo país contiene una 'l' separada de una 'n' por un único carácter y con cualquier número de caracteres por delante y por detrás de ese trío de letras (Finlandia y Holanda).

Nótese, de este último ejemplo, que los campos que se incluyen en una cláusula WHERE no tienen que ser los mismos que se incluyen en el resultado de la búsqueda.

Por último, el operador IN permite comprobar si un valor está en una "tupla".

```
SELECT Piloto, Puntos FROM Pilotos WHERE País IN ('Alemania', 'Inglaterra')
```

Question 5

Complete

Mark 2.00 out of 2.00

CompLete el siguiente código para recuperar los pilotos y las escuderías cuyos puntos están entre 100 y 200 de la tabla Pilotos de una base de datos SQLite3 desde Python:

```
sql = "SELECT Piloto, Escudería FROM Pilotos [ WHERE ] Puntos [ BETWEEN ] 100 AND 200"
cursor.execute(sql)
```

Information

La cláusula ORDER BY

Los resultados de una consulta se pueden ordenar usando la cláusula

... ORDER BY *columna1*, *columna2*, ...

Las opciones **ASC** o **DESC** tras cada columna significan, respectivamente, orden ascendente o descendente para la misma; pero si no se pone nada, se presupone ascendente.

Por ejemplo:

```
sql = "SELECT * FROM Pilotos WHERE Puntos > 150 ORDER BY Escudería DESC, Piloto"
cursor.execute(sql)
select_result = cursor.fetchall()
print(select_result)
```

```
[(3, 'Verstappen', 'Red Bull', 'Holanda', 181, 2), (2, 'Bottas', 'Mercedes', 'Finlandia', 188, 2), (1, 'Hamilton', 'Mercedes', 'Inglaterra', 250, 8)]
```

Question 6

Complete

Mark 2.00 out of 2.00

Complete el siguiente código para obtener los resultados con los pilotos de una misma escudería en orden alfabético inverso de sus nombres:

```
sql = "SELECT * FROM Pilotos WHERE Puntos > 150 [ ORDER ] BY Escudería, Piloto [ DESC ]"
cursor.execute(sql)
```

Information

Las instrucciones UPDATE y DELETE

Se puede *actualizar* uno (o más) registros —esto es, escribir en uno o más de sus campos— de una tabla usando la instrucción

UPDATE *tabla* SET *columna1* = *valor1*, *columna2* = *valor2*, ... WHERE *condición*

Por ejemplo:

```
UPDATE Pilotos SET Puntos = 300 WHERE Piloto = 'Hamilton';
```

La cláusula WHERE en la instrucción UPDATE es casi obligatoria; se puede omitir, pero, en ese caso, la actualización ¡afectará a todos los registros!, al no haber ningún filtro.

Lo mismo sucede en el caso de querer eliminar uno (o más) registros de alguna tabla, lo que se hace usando la instrucción

DELETE FROM *tabla* WHERE *condición*

Por ejemplo:

```
DELETE FROM Pilotos WHERE Escudería = 'Ferrari'
```

¡Olvidar la cláusula WHERE en una instrucción DELETE **vaciaría la tabla** en cuestión!

Question 7

Complete

Mark 3.00 out of 3.00

Complete el siguiente código para eliminar los registros de la tabla *Pilotos* en los que el valor del campo *Victorias* es igual a cero:

```
sql = "  DELETE  FROM Pilotos  WHERE  Victorias  = 0"
cursor.execute(sql)
```