



[Escuela de código]

Apuntes de clase:

Introducción a SQL

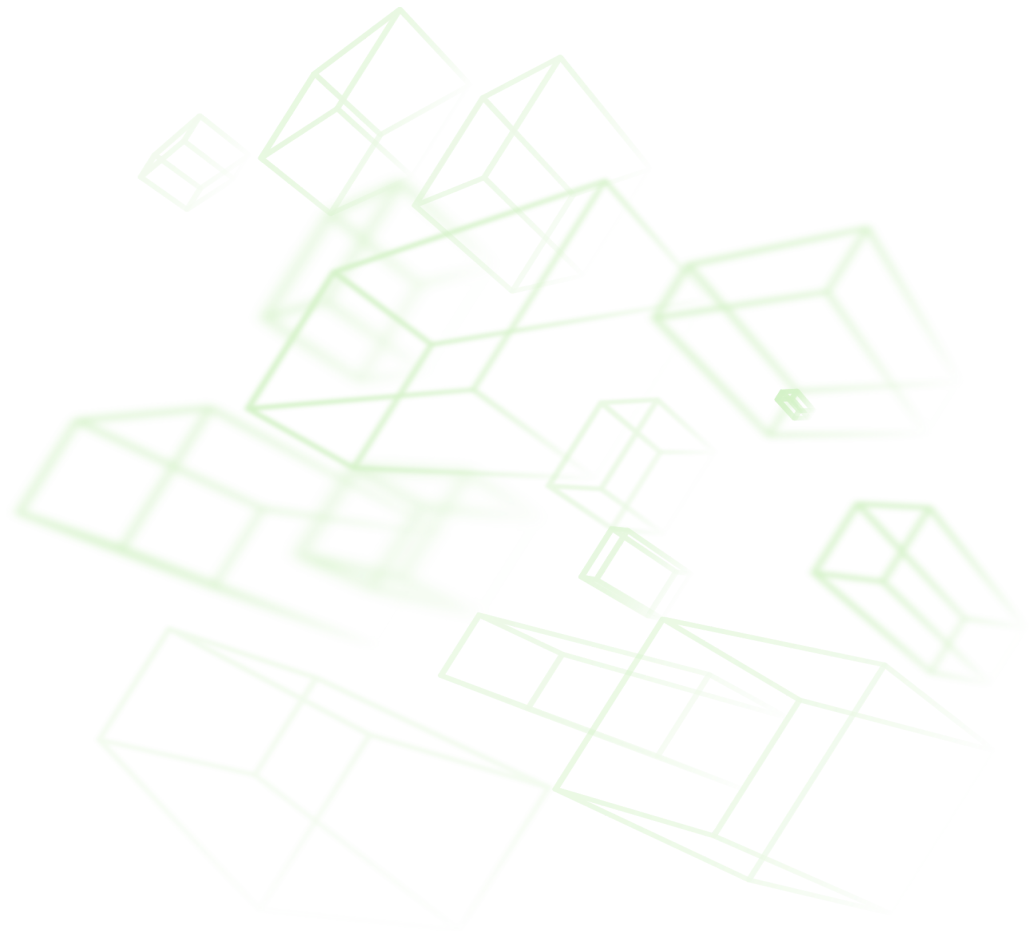


Tabla de contenido

Introducción a bases de datos	2
Base de datos	2
DBMS	2
Query	3
ORM - (Object-Relational mapping)	3
Ventajas de usar un sistema de base de datos (DBMS)	4
SQL - Structured Query Language	5
Esquema	5
UML	6
Atributos	6
Primary Key y Foreign Key	7
Primary key (PK)	7
Foreign Key (FK)	7
CRUD - Create Read Update Delete	8
Create	8
Tipos de datos	9
Restricciones (constraints)	10
Select	11
Insert	13
Update	14
Delete	16
Sintaxis de SQL	18
SQLite	19
Visualizadores online para practicar	19
Ejecución de queries en SQLite3	20
Consultar datos en SQLite	21
fetchall	21
fethone	21
Insertar datos en SQLite	22
Actualizar datos en SQLite	23
Borrar datos en SQLite	24
Insertar múltiples datos en SQLite	25
Descargar un repositorio de GitHub	26
Hasta la próxima!	26
Links de interés	26

Introducción a bases de datos

Base de datos

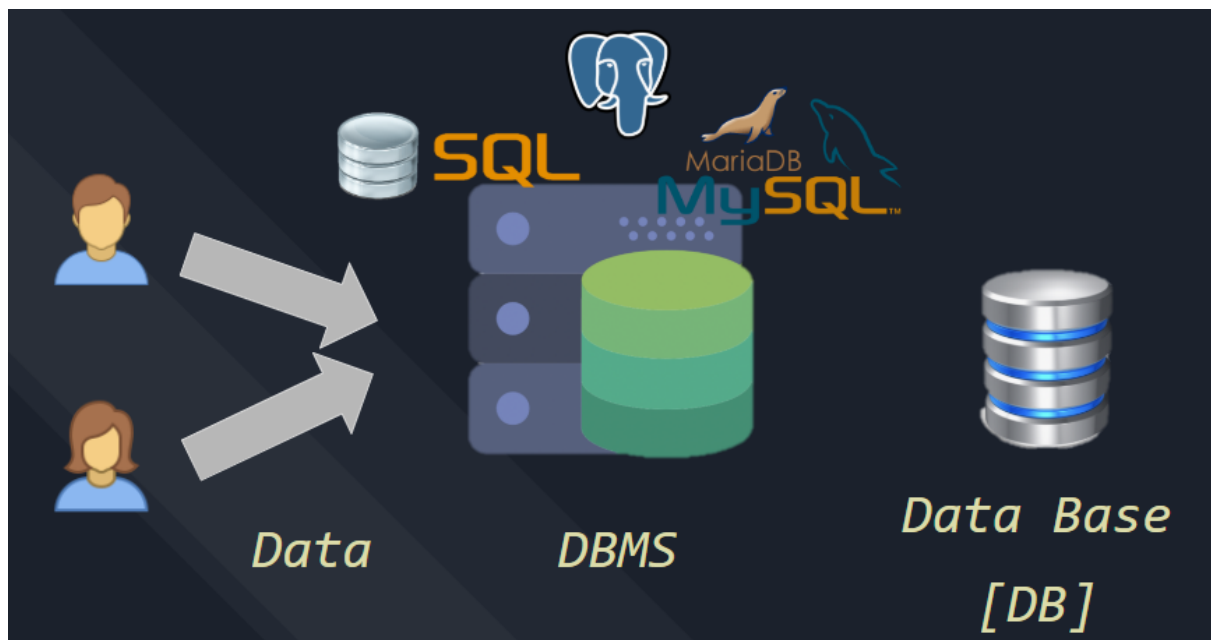
Una base de datos es una colección de información ordenada grabada en un disco duro de una PC. Está formada por información numérica, texto y alfanumérica. Una base de datos muy simple puede ser un archivo CSV, una planilla o un archivo de texto como lo conocemos.

DBMS

La mayoría de los sistemas interactúan con datos y esto lo realizan a través de un sistema de manejo de base de datos (DBMS - database management system).

El DBMS es un sistema que permite al usuario definir, crear, mantener y controlar el acceso a las bases de datos. SQL, MySQL, PostgreSQL son todos sistemas de administración y manejo de bases de datos (DBMS).

Antes de la existencia de los DBMS las bases de datos eran archivos de textos o CSV lo cuales eran muy difíciles de mantener y compartir al mismo tiempo entre diferentes usuarios.



Query

"query" o consulta es el medio o lenguaje estándar por el cual podemos enviar o traer datos a una tabla o base de datos. Está compuesta por la acción que se desea llevar a cabo (traer datos, actualizarlos, insertar nuevos, borrar) y los campos de datos.

ORM - (Object-Relational mapping)

En ORM permite utilizar distintos sistemas de bases de datos bajo una misma interfaz. Permite la reutilización y estandarización de código de lenguajes de bases de datos, generando una interfaz común. Esta interfaz nos abstrae de las diferencias que puede llegar a existir en el uso e implementación de por ejemplo MySQL, PostgreSQL, etc, que son todos sistemas basados en el lenguaje SQL para cada uno en algún punto tiene sus cuestiones especiales



El ORM más utilizado en Python es SqlAlchemy. En este curso no haremos uso del ORM ya que es preferible comenzar utilizando las queries crudas "raw" en el lenguaje nativo SQL que el día de mañana les permitirá entender otros sistemas SQL o utilizar cualquier ORM.

Además, hay un gran debate entre si es conveniente o no utilizar un ORM principalmente por posibles problemas de performance.

Ventajas de usar un sistema de base de datos (DBMS)

El método tradicional para guardar información es un archivo de texto o una planilla o CSV. El problema con este mecanismo son los siguientes:

- Integridad de los datos. No existe un marco o contexto para los datos, uno podría completar un campo de una columna con diferentes tipos de datos (numérico o texto) sin que nada lo evite.
- Un archivo es mucho más difícil de compartir de forma segura y concurrente entre otros usuarios.
- Agregar datos "en el medio" del contenido de un archivo, modificarlos o eliminarlos es una operación bastante compleja e ineficiente.



*Integridad
de datos*



*Acceso
controlado*



*Update y backup
eficiente*

IMPORTANTE: Utilizar un sistema de bases de datos para aplicaciones chicas suma mucha complejidad al proyecto, para aplicaciones pequeñas que capaz no tienen conectividad en la nube no es necesario un sistema completo de DBMS. Para estos casos es que existen módulos como SQLite que no implementan la parte del "servidor".

SQL - Structured Query Language

SQL (Structured Query Language) es un lenguaje estándar de bases de datos relacionales, del cual derivan sistemas de manejo relacionales de bases de datos (RDMBS) como PostgreSQL, MySQL, Oracle, Sybase, SQL Server, etc.

Esquema

Una base de datos SQL tiene la característica de tener un esquema relacional, a diferencia de las NoSQL como MongoDB que son "libre de esquema". Este esquema, si lo comparamos contra un archivo CSV, serían las columnas que posee ese archivo con sus respectivos nombres.

Veamos un ejemplo de lo que sería una tabla SQL de una persona:

id	name	age	nationality
1	Inove	12	Argentina
2	Python	29	Holanda
3	Max	35	Estados Unidos
4	Mirta	93	Argentina

Esta tabla tiene 4 columnas.

- id: Identificador único de esa persona (primary key).
- name: Nombre de la persona
- age: Edad de la persona
- nationality: Nacionalidad de la persona

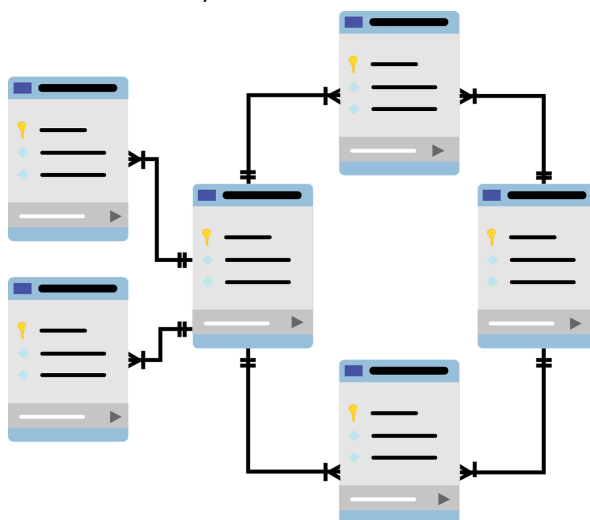
El hecho de que sea un sistema de base de datos relacional y no un archivo quiere decir que los campos de cada fila y columna pueden estar relacionados con otro campo de otra tabla. De esta forma se evita duplicar información y se agiliza el diseño del sistema. Por ejemplo, en nuestro esquema podría existir la tabla "nacionalidad" y completar la tabla anterior con las referencias a esta tabla:

id	name	age	nationality
1	Inove	12	1
2	Python	29	2
3	Max	35	3
4	Mirta	93	1

id	name
1	Argentina
2	Holanda
3	Estados Unidos

UML

Para representar el esquema SQL y poder ver las diferentes relaciones se utiliza un gráfico tipo UML. UML significa "Lenguaje Unificado de Modelo" y lleva ese nombre porque se utiliza no solo para modelar relaciones de bases de datos sino relaciones de código, programas y clases (programación orientada a objetos). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema



Atributos

id	name	age	nationality
1	Inove	12	1
2	Python	29	2
3	Max	35	3
4	Mirta	93	1

id	name
1	Argentina
2	Holanda
3	Estados Unidos

Los atributos son las columnas de cada tabla, y en cada caso, para cada fila, el atributo tomará un valor determinado (numérico, texto, alfanumérico o vacío "null"). Para definir un atributo se debe especificar su nombre, su tipo básico de datos

- **Atributo simple:** Podría ser cualquier tipo de dato como por ejemplo en este caso "name" o "age" que no tenga relación con otra tabla.
- **Atributo compuesto:** Este atributo tiene la dirección o vínculo con otra tabla donde se encuentra mayor información detallada del atributo en sí (relación). Un ejemplo en este caso es nacionalidad, que el atributo compuesto es la referencia la tabla nacionalidad donde no solo podría encontrar el nombre de esa nacionalidad sino también datos del país en general (idioma oficial, etc).

Primary Key y Foreign Key



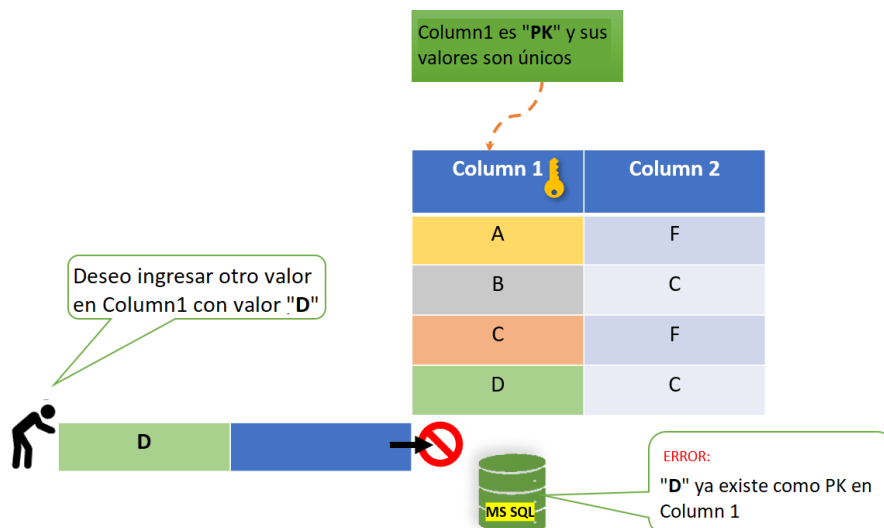
id	name	age	nationality
1	Inove	12	1
2	Python	29	2
3	Max	35	3
4	Mirta	93	1

id	name
1	Argentina
2	Holanda
3	Estados Unidos

Las llaves funcionan como identificador único de una fila en la tabla de la base de datos, como puede ser el número de fila (ID), el DNI de una persona, la fecha en la que se ingresó esa fila, etc.

Primary key (PK)

Cuando un atributo funciona como identificador único de esa entrada o fila al atributo se lo llama "clave primaria" (primary key). Este valor se utiliza como identificador, es decir, que el sistema utiliza ese valor para diferenciar esa fila de otras en la tabla. El identificador no puede repetirse, ya que es unívoco por fila. Veamos un ejemplo de caso de falla:



Foreign Key (FK)

Cuando una entidad tiene un atributo como clave de entidad ajena se la llama clave ajena (foreign).

CRUD - Create Read Update Delete

Todas las sentencias que construyamos de aquí en más siguen las siguientes normas:

- Todo lo relativo a SQL (sentencias SQL) se coloca en mayúscula, para poder identificar de golpe de vista que es sintaxis SQL y que son datos.
- Todas las sentencias SQL deben terminar con punto y coma ";"

Create

"Create" es la instrucción SQL que nos permitirá el día de mañana crear nuestras tablas. El ejemplo más simplificado del uso del comando es el siguiente:

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
);
```

En este caso estamos creando una tabla "table_name" que posee tres atributos o columnas con los nombres column1, column2 y column3 los cuales llevan un tipo de dato específico

Hay que tener en cuenta que no podemos crear una tabla SQL si ésta ya existe. Si por alguna razón queremos volver a crear la tabla (por ejemplo para deshacernos de forma rápida de todo su contenido y construirla con un nuevo esquema) debemos primero eliminar la tabla:

```
DROP TABLE IF EXISTS table_name;
```

Tipos de datos

SQL tiene una cantidad inimaginable de tipos de datos los cuales se dividen en:

- Datos numéricos.
- Datos de texto.
- Datos del tipo tiempo.

Existen diferentes tipos de datos para cada uno en función de la precisión o cantidad de información (espacio de memoria) que queramos almacenar. Nosotros para mantenerlo lo más acotado utilizaremos:

Tipo	Sintaxis SQL	Ejemplo
Enteros	INTEGER	Cualquier número sin coma → 1250
Decimales	DECIMAL	Todos los números con coma → 5.6
Texto	TEXT	Cualquier texto → "Hola"
Tiempo	DATETIME	"2019-06-05 19:50:01"

Veamos el ejemplo que desarrollaremos en clase:

```
CREATE TABLE persona(  
    [id] INTEGER PRIMARY KEY AUTOINCREMENT,  
    [name] TEXT NOT NULL,  
    [age] INTEGER,  
    [nationality] TEXT  
);
```

Dos cosas muy importantes para mencionar del ejemplo anterior:

- Los nombres de las columnas están colocados entre corchetes "[]", no es necesario que respeten esa sintaxis pero es más legible y rápido de identificar los nombres.
- Todo lo relativo a SQL (sentencias SQL) se coloca en mayúscula, por la misma razón que antes es para poder identificar de golpe de vista que es sintaxis SQL y que son datos.
- El "id" tiene unas sentencias SQL especiales (PRIMARY KEY, AUTOINCREMENT). Estos elementos son "constraints", es decir, restricciones o configuraciones de la tabla.

Restricciones (constraints)

Veamos un ejemplo más completo de como crear una tabla SQL

```
CREATE TABLE persona(  
    [id] INTEGER PRIMARY KEY AUTOINCREMENT,  
    [name] STRING NOT NULL,  
    [age] INTEGER NOT NULL,  
    [nationality] INTEGER NOT NULL REFERENCES nacionalidad(id)  
);
```

En este ejemplo aparecen muchos otros "constraints", estos serán todos los tipos de restricciones que abordaremos en esta instancia introductoria:

Constraint	Descripción
PRIMARY KEY	Declarar a la columna como "clave primaria"
REFERENCES	Declarar una columna como "clave ajena" especificando la tabla y columna para la cual se crea la relación
AUTOINCREMENT	Se utiliza para auto incrementar el valor de nuevos registros de claves primarias. Si un nuevo registro en una tabla queremos que la clave primaria (por ejemplo el ID) valga el valor siguiente al ultimo ID ingresado en esa tabla, con la sentencia AUTOINCREMENT se realiza automáticamente.
NOT NULL	Se utiliza para forzar de que esa columna sea completada si o si cuando se crea un nuevo registro en la tabla

Select

"SELECT" es la sentencia que más utilizaremos en SQL, y se utiliza para buscar y extraer información de la base de datos (es como un find). Al SELECT podemos especificarle de qué tabla queremos extraer el contenido (FROM), qué columnas deseamos obtener y filtrar por algún criterio si deseamos (WHERE).

```
SELECT column1, column2, ... FROM table_name;
```

El caso más usual es donde deseamos obtener todas las columnas y todas las filas de una determinada tabla:

```
SELECT * FROM table_name';
```

En este caso estamos indicando que deseamos leer todas las columnas (por eso el asterisco) de la tabla "tabla_name". Además, como no especificamos ninguna condiciones esa operación devolverá todas las filas que existan en la base de datos.

Si deseamos agregar una condición (WHERE) para que el sistema no devuelva todas las filas disponibles de la tabla:

```
SELECT column1, column2, ... FROM table_name WHERE condition;
```

Veamos algunos ejemplos utilizando la misma tabla "persona" que utilizamos hasta el momento:

1 - Solicitar todas las columnas y todas las filas de la tabla persona



```
SELECT * FROM persona;
```

id	name	age	nationality
1	Inove	12	Argentina
2	Python	29	Holanda
3	Max	35	Estados Unidos
4	Mirta	93	Argentina

Este es uno de los casos más comunes de uso, se utiliza fundamentalmente para conocer todos los datos almacenados en una tabla hasta el momento.

2 - Solicitar todas las columnas y las filas de persona cuya nacionalidad sean Argentina

```
SELECT * FROM persona
WHERE nationality = 'Argentina';
```

id	name	age	nationality
1	Inove	12	Argentina
2	Python	29	Holanda
3	Max	35	Estados Unidos
4	Mirta	93	Argentina

Este es el caso típico de filtrado de tabla, en donde solo pediremos la sistema las filas que nos interesa.

3 - Solicitar las columnas de "name" y "age" de las filas de persona cuya nacionalidad sean Argentina

```
SELECT name, age FROM persona
WHERE nationality = 'Argentina';
```

id	name	age	nationality
1	Inove	12	Argentina
2	Python	29	Holanda
3	Max	35	Estados Unidos
4	Mirta	93	Argentina

Normalmente las tablas tienen muchas columnas y no siempre nos interesa todo su contenido, es por eso que también podemos solicitar solo aquellas columnas que nos sean de interés.

Insert

"INSERT" podría decirse que es la segunda sentencia más utilizada en SQL, se utiliza para agregar nueva información a la tabla. En este proceso debemos declarar en qué tabla queremos insertar los datos y que columnas completamos de esa tabla, ya que no necesariamente tenemos la obligación de completar los datos para todas las columnas.

IMPORTANTE: Estamos obligados a completar con INSERT si o si aquellas columnas marcadas como "NOT NULL" ya que no estas no aceptan estar vacías.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Cuando realizamos un "INSERT" podemos solo especificar las columnas que deseamos completar en la inserción del registro o fila en la tabla, siempre y cuando completamos mínimamente aquellas columnas esenciales (las marcadas como NOT NULL). En el ejemplo de la tabla "persona" las únicas columna esenciales son el id (por que es una clave primara) y el nombre "name":

```
INSERT INTO persona (name)
VALUES ('SQL');
```

```
INSERT INTO persona (name, age,
nationality)
VALUES ('SQL', 46, 'Inglaterra');
```

id	name	age	nationality
1	Inove	12	Argentina
2	Python	29	Holanda
3	Max	35	Estados Unidos
4	Mirta	93	Argentina
5	SQL		

id	name	age	nationality
1	Inove	12	Argentina
2	Python	29	Holanda
3	Max	35	Estados Unidos
4	Mirta	93	Argentina
5	SQL	46	Inglaterra

- Como podemos ver en ambos ejemplos no se especifico y completó la columna "id" que era la clave primaria porque esta si recuerdan se marcó como "AUTOINCREMENT", por lo que a pesar de no especificar el valor deseado el sistema automáticamente tomó el valor anterior de la fila anterior y lo incrementó.
- En el primer ejemplo se ve que solo se completa la columna esencial "name".
- En el segundo caso se completan todas las columnas de la tabla (excepto por "id" ya que no es necesario en este caso).

Update

La sentencia "UPDATE" no es muy utilizada ya que en general siempre agregamos información nueva como ya se mostró con el "INSERT", pero "UPDATE" nos permite actualizar una o más filas de una tabla que cumplan determinado criterio y actualizar una o varias columnas.

```
UPDATE table_name SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

Podemos editar una columna o varias en una sentencia de "UPDATE", y podemos editar una o todas las filas de la tabla.

```
UPDATE table_name SET column1 = value1;
```

Se puede agregar una condición (WHERE) para editar solo una o algunas filas específicas según la condición:

```
UPDATE table_name SET column1 = value1 WHERE condition;
```

Veamos algunos ejemplos utilizando la misma tabla "persona" que utilizamos hasta el momento:

1 - Actualizar la columna "name" en todas las filas de la tabla



```
UPDATE persona SET age = 1;
```

id	name	age	nationality
1	Inove	1	Argentina
2	Python	1	Holanda
3	Max	1	Estados Unidos
4	Mirta	1	Argentina

No es muy común este caso de uso ya que generalmente se actualiza solo algunas filas de la tabla.

2 - Actualizar la columna "name" en las filas de la tabla de persona cuya nacionalidad sean Argentina

```
UPDATE persona SET age = 1  
WHERE nationality = 'Argentina';
```

id	name	age	nationality
1	Inove	1	Argentina
2	Python	1	Holanda
3	Max	1	Estados Unidos
4	Mirta	1	Argentina

Este es uno de los casos típicos de edición o actualización de los campos de una tabla.

3 - Actualizar la columna "name" en la fila N° 3 (id=3)

```
UPDATE persona SET age = 1  
WHERE id = 3;
```

id	name	age	nationality
1	Inove	12	Argentina
2	Python	29	Holanda
3	Max	1	Estados Unidos
4	Mirta	93	Argentina

Este es el caso más usual, en donde solo se edita una fila de la tabla y a esta se la distingue por la clave primaria que la identifica (en este caso el id).

Delete

La sentencia "DELETE" es muy extraño que se utilice, ya que los datos siempre serán de utilidad, normalmente en vez de borrarlos de la base de datos se los invalidan (se marca como que los datos son obsoletos o no deben mostrarse, se filtran).


De todas formas es una sentencia más cuyo comportamiento es muy similar al de UPDATE:

```
DELETE FROM table_name WHERE condition;
```

El condicional (WHERE) se utiliza para determinar qué filas se desean eliminar de la tabla, en caso de no especificarse la condición se borrarán todas las filas:

```
DELETE FROM table_name;
```

1 - Borrar todas las filas de la tabla persona



The image shows a SQL query editor with the command `DELETE FROM persona;` and a table named 'persona' with four rows. Each row is crossed out with a red line, indicating that all data has been deleted.

id	name	age	nationality
1	Inove	12	Argentina
2	Python	29	Holanda
3	Max	33	Estados Unidos
4	Mirta	33	Argentina

No es muy común este caso de uso, pero en caso de desear borrar todos los elementos de una tabla es una forma de lograrlo.

2 - Borrar todas las filas de la tabla persona cuya nacionalidad sea 'Argentina'

```
DELETE FROM persona  
WHERE nationality = 'Argentina';
```

id	name	age	nationality
1	Inove	12	Argentina
2	Python	29	Holanda
3	Max	35	Estados Unidos
4	Mirta	93	Argentina

Este es uno de los casos típicos donde se eliminan aquellas columnas que cumplan con cierta condición.

3 - Borrar la fila Nº 3 (id=3)

```
DELETE FROM persona  
WHERE id = 3;
```

id	name	age	nationality
1	Inove	12	Argentina
2	Python	29	Holanda
3	Max	35	Estados Unidos
4	Mirta	93	Argentina

Este es el caso más usual, en donde solo se borra una fila de la tabla y a esta se la distingue por la clave primaria que la identifica (en este caso el id).

Sintaxis de SQL

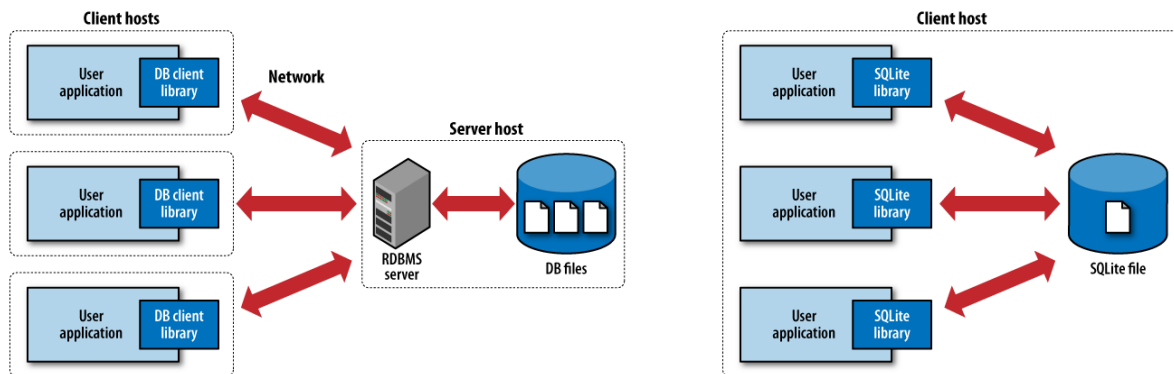
Lo más importante a tener en cuenta a la hora de escribir tablas y columnas SQL son algunas normas básicas de sintaxis muy parecidas a las que venimos usando para Python:

- Lo nombres de las columnas deben ser lo más descriptivos posibles pero sin excederse, escritos en letra minúscula.. En caso de requerir más de dos palabras usar guión bajo para separarlas.
- Los nombres de las tablas deben englobar o describir los datos que se almacenará en ella, y debe estar escrito en singular. Por ejemplo, para una tabla donde almaceno datos de varias personas la tabla se llamará "persona", ya que el esquema corresponde a los datos que debe ingresar cada persona.
- Los strings se arman con simple comilla → 'Inove'.

Para más información les dejamos una guía extensa de uso y buenas prácticas de SQL para que tengan en consideración para proyectos más grandes:

SQLite

Sqlite sigue siendo un sistema de bases de datos relacional (RDBMS) pero no es "servidor-client", en la red sino que solo existe de forma local para el cliente:



SQLite viene integrado junto con los módulos estándar de Python y se invoca como:

```
import sqlite3
```

IMPORTANTE A TENER EN CUENTA: En SQLite se usa "dynamic typing", esto quiere decir que una columna que definimos como "INTEGER" (números) acepta texto (TEXT) y el sistema no devolverá ningún tipo de error y se almacenará ese dato tipo texto en la columna para datos numéricos.

Documentación oficial de Python SQLite3

<https://docs.python.org/3/library/sqlite3.html>

Visualizadores online para practicar

<https://extendsclass.com/sqlite-browser.html>

<https://inloop.github.io/sqlite-viewer/>

Ejecución de queries en SQLite3

El primero paso es conectarse a la base de datos (DB), en este caso como la base de datos se trata de un archivo es conectarse al archivo que en caso de no existir se genera automáticamente:

```
# Conectarnos a la base de datos
# En caso de que no exista el archivo se genera
# como una base de datos vacía
conn = sqlite3.connect('personas.db')
```

El siguiente punto es generar un cursor, el cursor lo utilizamos para ejecutar queries o levantar datos de las consultas:

```
# Crear el cursor para poder ejecutar las queries
c = conn.cursor()
```

De ahora en más podemos ejecutar cualquier tipo de query, como por ejemplo crear una tabla "persona" como vimos en los ejemplos:

```
# Ejecutar una query
c.execute("""
    CREATE TABLE persona(
        [id] INTEGER PRIMARY KEY AUTOINCREMENT,
        [name] TEXT NOT NULL,
        [age] INTEGER,
        [nationality] TEXT
    );
""")
```

Es importante que luego de terminar de ejecutar las queries deseadas en un bloque de código se realice "commit" para que los cambios efectuados en la DB surtan efecto en el archivo (es como el flush de archivos):

```
# Para salvar los cambios realizados en la DB debemos
# ejecutar el commit, NO olvidarse de este paso!
conn.commit()
```

Por último debemos cerrar la conexión cuando no realizaremos más cambios:

```
# Cerrar la conexión con la base de datos
conn.close()
```

Consultar datos en SQLite

La consulta a la base de datos se realiza a través de la ejecución de la sentencia SQL "SELECT", pero los datos no serán extraídos en ese momento de la base de datos sino que los tendremos en un cursor a nuestra disposición para que los retiremos todos juntos o de a uno según nuestra conveniencia.

Lo primero es realizar la consulta a la base de datos, en este ejemplo deseamos obtener todas las columnas y filas de la tabla "persona" que vimos hasta el momento:

```
# Conectarse a la base de datos
conn = sqlite3.connect('personas.db')
c = conn.cursor()
c.execute('SELECT * FROM persona')
```

fetchall

Utilizaremos este método de SQLite para retirar todos los datos de la query realizada:

```
data = c.fetchall()
print(data)
```

En trabajos grandes donde las bases de datos tienen más información que la capacidad de nuestra memoria RAM no se aconseja retirar todos los datos de una con "fetchall" sino es que conveniente retirarlos de a uno con "fetchone" para no ocupar toda la memoria del sistema.

fetchone

Utilizaremos este método de SQLite para retirar los datos uno a uno de la query realizada:

```
print('Recorrer los datos desde el cursor')
while True:
    row = c.fetchone()
    if row is None:
        break
    print(row)
```

Cuando no haya más datos por leer fetchone retorna "None"

Insertar datos en SQLite

Agregar datos a la base de datos se realiza a través de la sentencia "INSERT". La ventaja de utilizar sqlite3 es que el método "execute" que ya se puso a prueba en los puntos anteriores puede recibir como segundo parámetros los datos que se desean "bindear" (unir) a la query:

```
c.execute(query, values)
```

El primer parámetro de la execute debe ser la query y en segundo lugar deben ir los valor. Si se deseara se podría realizar todo directo con query pero estaría forzando datos en el texto que pueden cambiar el día de mañana:

```
c.execute(""" INSERT INTO persona (name, age, nationality)
VALUES ('Inove',12,'Argentina');""")
```

En vez de escribir los datos en la query (en el texto) utilizamos la propiedad de "bind" que posee SQL para añadir datos a una query, esto se realiza con el símbolo "?". Es funcionamiento es muy similar a como se viene utilizando las llaves "{}" para los textos, el sistema los datos pasados como parámetros y los ubica uno a uno en todos los "?" encontrados en la query:

```
c.execute("""
INSERT INTO persona (name, age, nationality)
VALUES (?, ?, ?);""", ('Inove', 12, 'Argentina'))
```

Para mayor prolijidad se puede armar una variable aparte con los datos que se desean bindear en una lista o tupla:

```
values = ('Inove', 12, 'Argentina')
c.execute("""
INSERT INTO persona (name, age, nationality)
VALUES (?, ?, ?);""", values)
```

En estos ejemplos hay 3 datos que se pasan por parámetro (Inove, 12 y Argentina) y hay 3 "?" en la query, los datos se completan en el orden que se pasan como parámetros.

Actualizar datos en SQLite

Para actualizar la tabla de la bases de datos utilizaremos la sentencia "UPDATE". Utilizaremos el concepto de "bind" para gestionar el criterio de búsqueda (WHERE) y los datos que se desean actualizar.

En este ejemplo veremos cómo actualizar una fila de la tabla "persona" que vimos hasta ahora donde deseamos actualizar la edad de las personas llamadas "Max":

```
# Conectarse a la base de datos
conn = sqlite3.connect('personas.db')
c = conn.cursor()
rowcount = c.execute("UPDATE persona SET age =? WHERE name =?",
                      (age, name)).rowcount

print('Filas actualizadas:', rowcount)

# Save
conn.commit()
# Cerrar la conexión con la base de datos
conn.close()
```

En este caso se puede ver que se aprovecha el uso de binding con "?" para pasar a la query los valores de la edad de la persona y el nombre (almacenados en las variables age y name).

Otro detalle es que se puede llamar al atributo "rowcount" de la query ejecutada para tener una noción de la cantidad de filas actualizadas.

IMPORTANTE: Como se está actualizando el contenido de la tabla se debe llamar a "commit".

Borrar datos en SQLite

Para eliminar una fila de tabla de la bases de datos utilizaremos la sentencia "DELETE". Utilizaremos el concepto de "bind" para gestionar el criterio de búsqueda (WHERE) para determinar las filas que se desean eliminar

En este ejemplo veremos cómo eliminar una fila de la tabla "persona" que vimos hasta ahora cuyas filas posea el nombre "Max":

```
# Conectarse a la base de datos
conn = sqlite3.connect('personas.db')
c = conn.cursor()

# Borrar la fila cuyo nombre coincida con la búsqueda
# NOTA: Recordar que las tupla un solo elemento se definen como
# (elemento,)
# Si presta a confusión usar una lista --> [elemento]
rowcount = c.execute("DELETE FROM persona WHERE name =?", (name,)).rowcount

print('Filas actualizadas:', rowcount)

# Save
conn.commit()
# Cerrar la conexión con la base de datos
conn.close()
```

En este caso se está pasando como valor un único parámetro a la query, en caso de utilizar tuplas se debe recordar que una tupla de un solo parámetro se define como: (valor,) → se debe agregar la coma para que el sistema sepa que se trata de una tupla.

Otro detalle es que se puede llamar al atributo "rowcount" de la query ejecutada para tener una noción de la cantidad de filas actualizadas.

IMPORTANTE: Como se está actualizando el contenido de la tabla se debe llamar a "commit".

Insertar múltiples datos en SQLite

Para agregar múltiples datos a la base de datos se realiza a través de la sentencia "INSERT" de la misma forma que antes pero ahora se utiliza el método "**executemany**" de SQLite junto con el "bind" de variables:

```
conn = sqlite3.connect('personas.db')
c = conn.cursor()

group = [('Max', 40, 'Estados Unidos'),
         ('SQL', 13, 'Inglaterra'),
         ('SQLite', 20, 'Estados Unidos'),
         ]

c.executemany("""
    INSERT INTO persona (name, age, nationality)
    VALUES (?, ?, ?);""", group)

conn.commit()
# Cerrar la conexión con la base de datos
conn.close()
```

La diferencia es que ahora se pasa como parámetros una lista de listas o una lista de tuplas, donde cada lista o tupla interna representan los datos que se desean agregar por fila.

Primera fila que se agrega → ('Max', 40, 'Estados Unidos')

Segunda fila que se agrega → ('SQL', 13, 'Inglaterra')

Tercera fila que se agrega → ('SQLite', 20, 'Estados Unidos')

Descargar un repositorio de GitHub

Para poder realizar las actividades de aquí en adelante debemos tener instalado y configurado nuestro GitHub. Todos los ejemplos prácticos estarán subidos al repositorio GitHub de **InoveAlumnos**, para aprender como descargar estos ejemplos desde el repositorio referirse al "Instructivo de GitHub: Descargar un repositorio" disponible entre los archivos del campus. De no encontrarse allí, por favor, tenga a bien comunicarse con alumnos@inove.com.ar para su solicitud.

Debemos descargar el repositorio que contiene los ejemplos de clase de ésta unidad:

https://github.com/InoveAlumnos/sql_intro_python

Hasta la próxima!

Con esto finaliza el tema "SQL Introducción", a partir de ahora tienen las herramientas mínimas para poder comenzar a utilizar una base de datos SQL.

Si desean conocer más detalles sobre el contenido pueden iniciar un tema de discusión en el foro del campus, o visitar los "Links de interés" que se encuentran al final de este apunte.

Links de interés

- [Documentación oficial Python SQLite](#)
- [Ejemplos de uso Sqlite](#)
- [Similitudes entre Sqlite, Postgre y MySQL](#)
- [ORM de Python](#)
- [Formato de instrucciones SQL](#)
- [Creación de tablas](#)
- [Coding Style Guide](#)
- [Libro Fundamentos SQL - Andy Oppel y Robert Sheldon](#)
- [Visualizador online 1](#)
- [Visualizador online 2](#)
- [Ejemplos con CREATE](#)
- [Ejemplos con SELECT](#)
- [Ejemplos con INSERT](#)
- [Ejemplos con UPDATE](#)
- [Ejemplos con DELETE](#)