



Fundamentos Tecnologías Emergentes con Python

Fundamentos Tecnologías Emergentes con Python



PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL
FORMATO GUÍA DE APRENDIZAJE
FUNDAMENTOS DE TECNOLOGÍAS EMERGENTES (PYTHON)

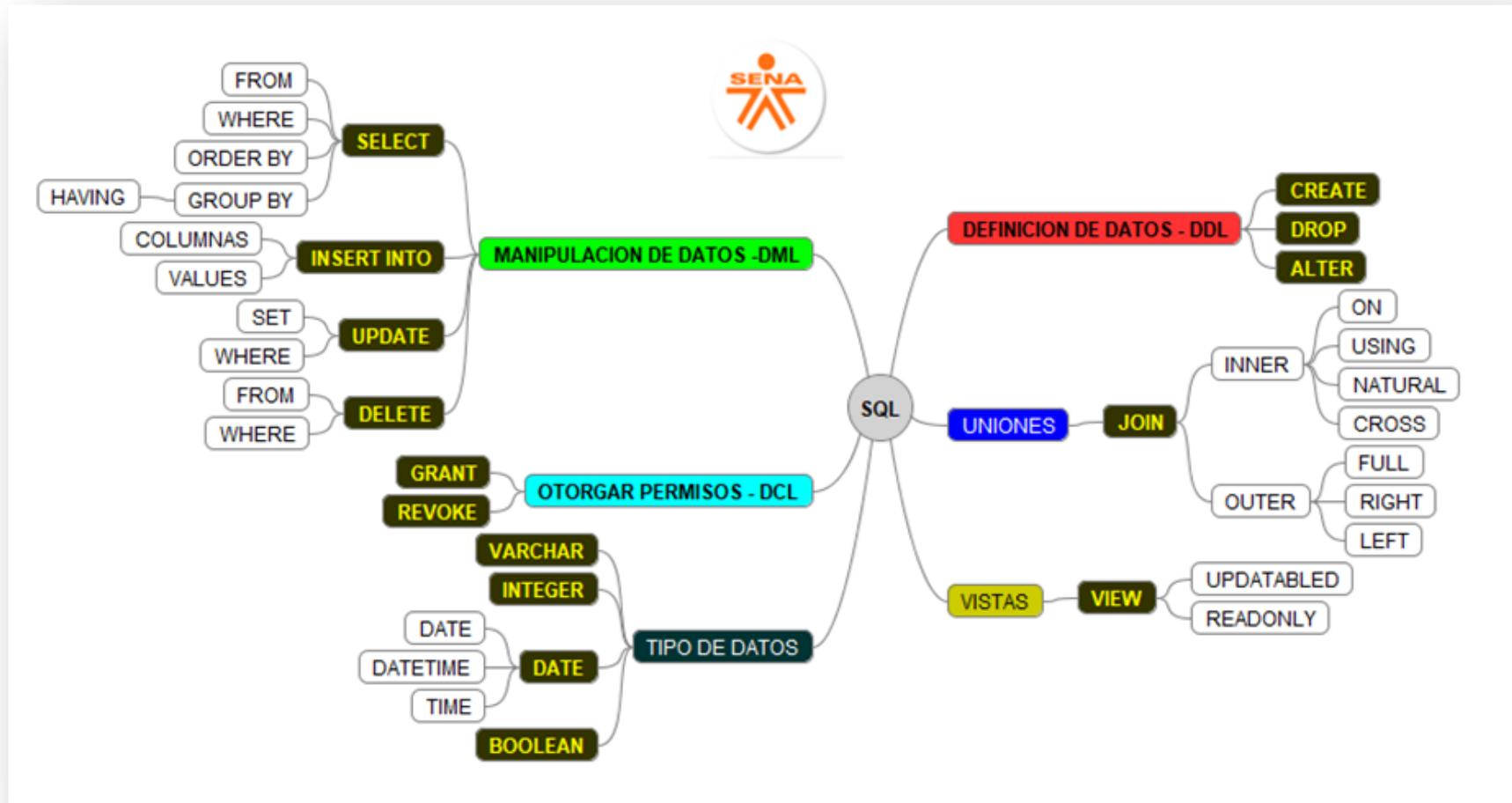
IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: ANALISIS Y DESARROLLO DE SOFTWARE.
- Código del Programa de Formación: 228118
- Nombre del Proyecto: 2605752 DESARROLLO DE SOFTWARE PARA EL CONTROL DE PROCESOS ORIENTADOS A MICROSERVICIOS.
- Fase del Proyecto: EVALUACIÓN
- Actividad de Proyecto: REALIZAR ACTIVIDADES DE VERIFICACIÓN DE CALIDAD DEL SOFTWARE
- Competencia: 220501098 - ADOPCIÓN DE BUENAS PRÁCTICAS EN EL PROCESO DE DESARROLLO DE SOFTWARE.
- Resultados de Aprendizaje Alcanzar:
- 220501098 02 VERIFICAR LA CALIDAD DEL SOFTWARE DE ACUERDO CON LAS PRÁCTICAS ASOCIADAS EN LOS PROCESOS DE DESARROLLO.
- Duración de la Guía: 66 Horas (DIRECTO + INDEPENDIENTE)

Conectividad a Base de Datos

jgalindos@sena.edu.co 2023

Lenguaje de Consulta estructurada



Conectando al servidor MySQL

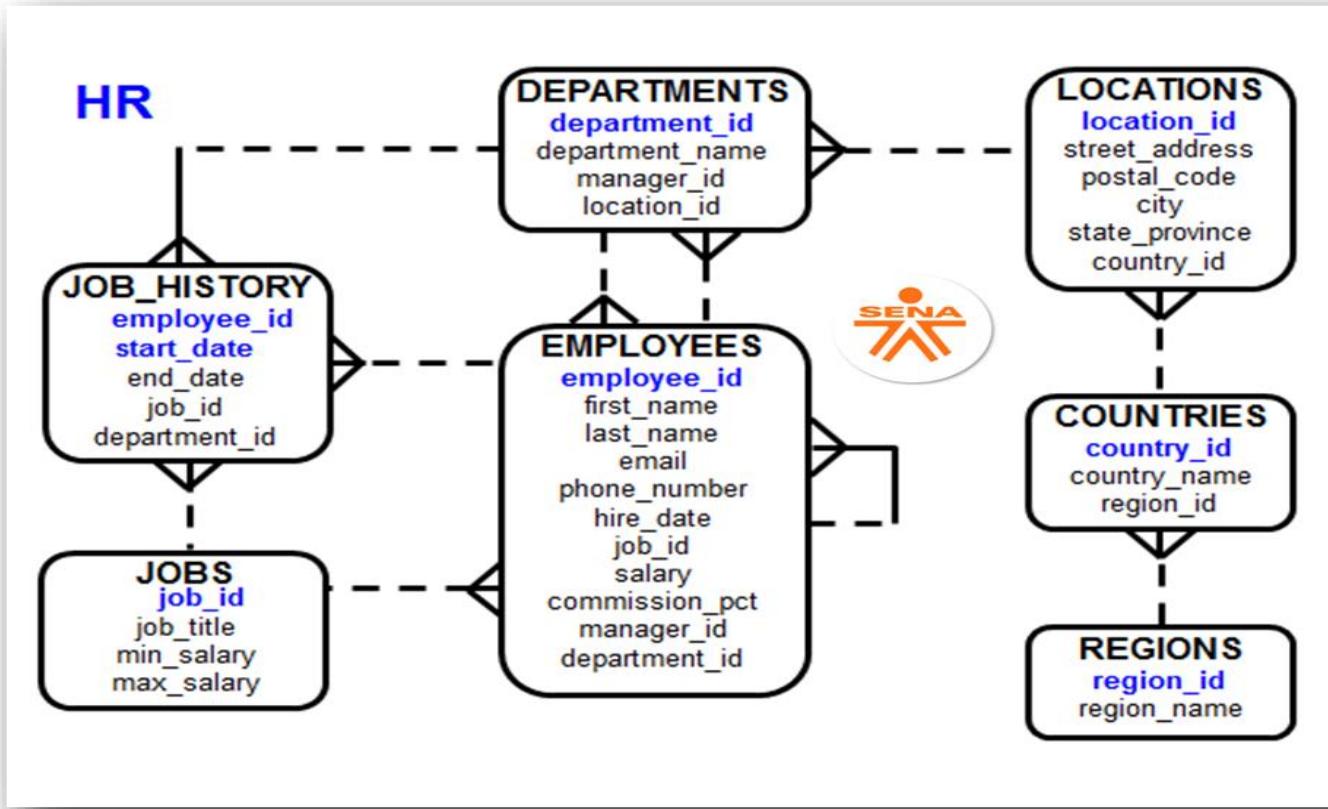


```
# mysql -u root -p
Enter password: ****
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 12
Server version: 10.1.37-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Crear el esquema de base de datos



- Descargar el script de creación de la base de datos [HR](#)
- Copiar el script al block de notas.

```
422 líneas (402 sloc) | 24.2 KB
1 ...
2 -- Host:          127.0.0.1
3 -- Versión del servidor: 10.1.37-MariaDB - mariadb.org binary distribution
4 -- SO del servidor: Win32
5 -- HeidiSQL Versión: 10.2.0.5599
6 ...
7
8 /*!40101 SET OLD_CHARACTER_SET_CLIENT=@CHARACTER_SET_CLIENT */;
9 /*!40101 SET NAMES utf8 */;
10 /*!50503 SET NAMES utf8mb4 */;
```

- Pegar desde el portapapeles a la consola Mysql

```
Mariadb [hr]> /*!40101 SET SQL_MODE=IFNULL(@OLD_SQL_MODE, '') */;
Query OK, 0 rows affected (0.00 sec)
Mariadb [hr]> /*!40014 SET FOREIGN_KEY_CHECKS=IF(@OLD_FOREIGN_KEY_CHECKS IS NULL, 1, @OLD_FOREIGN_KEY_CHECKS) */;
Query OK, 0 rows affected (0.00 sec)
Mariadb [hr]> /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
Query OK, 0 rows affected (0.00 sec)
Mariadb [hr]> \c
C:\Users\...
Mariadb [hr]> show tables;
+-----+
| Tables_in_hr |
+-----+
| countries   |
| emp_details |
| emp_details_view |
| employees   |
| jobs        |
| locations   |
| regions     |
+-----+
8 rows in set (0.00 sec)

Mariadb [hr]> _
```

Conectando MySQL Python



```
pip install mysql-connector-python
```

Importar las bibliotecas necesarias

```
import mysql.connector  
import pandas as pd
```

Realizar la conexión hacia el servidor mysql

```
conn=mysql.connector.connect(  
    host="localhost",  
    user="root",  
    passwd="",  
    database="hr",  
    port=3306  
)
```

Conectando MySQL Python



Crear un cursor para interactuar con MySQL

```
cursor=conn.cursor()
```

Obtener todas las filas de la tabla employees

```
sq="select * from employees"  
cursor.execute(sq)  
resultado=pd.DataFrame(cursor.fetchall())  
print(resultado)
```

PRACTICA MIGRANDO DESDE EXCEL CON PYTHON



Instalar los siguientes drivers y descargar el archivo [Neptuno.xls](#):

```
#pip install mysql-connector-python  
#pip install xlrd
```

Importar las bibliotecas necesarias

```
import mysql.connector  
import xlrd  
import pandas as pd
```

Realizar la conexión hacia el servidor mysql

```
conn=mysql.connector.connect(  
    host="localhost",  
    user="root",  
    passwd="",  
    database="neptuno",  
    port=3306  
)
```

PRACTICA MIGRANDO DESDE EXCEL CON PYTHON



Crear un cursor para interactuar con MySQL

```
cursor=conn.cursor()
```

Crear la tabla si no existe

```
sq="CREATE TABLE IF NOT EXISTS CATEGORIA(  
    IDCATEGORIA INTEGER PRIMARY KEY, NOMBRECATEGORIA TEXT, DESCRIPCION TEXT  
)"
```

Ejecutar la consulta

```
cursor.execute(sq)  
conn.commit()
```

PRACTICA MIGRANDO DESDE EXCEL CON PYTHON



INSERTAR REGISTROS EN LA TABLA CATEGORIAS

Cargar la hoja “2,-Categorias” del archivo Excel “Neptuno.xls”

```
x=pd.read_excel("c:/Borrar/Neptuno.xls",sheet_name="2,-Categorias")
```

Se cargan los nombres de las columnas

```
k=[]
for i in x:
    k.append(i)
```

```
z0=x[k[0]]
z1=x[k[1]]
z2=x[k[2]]
```

Cantidad de filas

```
j=len(x)
```

PRACTICA MIGRANDO DESDE EXCEL CON PYTHON



INSERTAR REGISTROS EN LA TABLA CATEGORIAS

Recorrer las filas para construir la instrucción insert y se ejecuta para crear el registro”

```
i=0
while i<j:
    sql="insert      into      Categorias(idcategoria,NombreCategoria,Descripcion)
values(%s,%s,%s)"
    sql1=(int(z0[i]),z1[i],z2[i])
    print(sql1)
    i=i+1
    cursor.execute(sql,sql1)
    connection.commit()
```

Se cierra la conexión al servidor

```
connection.close()
```

CONECTANDO MONGODB CON PYTHON



CONECTANDO MONGODB CON PYTHON

CONECTANDO MONGODB CON PYTHON



Conectando Python con MongoDB

Conectar al servidor MongoDB local



```
import pymongo
from pymongo import MongoClient

CONNECTION_STRING = "mongodb://localhost:27017/"

# Create a connection using MongoClient. You can import MongoClient or use pymongo.MongoClient
client = MongoClient(CONNECTION_STRING)
```

Crear la base de datos llamada “titulada” y la colección “aprendices”

```
dbname = client["titulada"]
colección=dbname["aprendices"]
```

CONECTANDO MONGODB CON PYTHON



Conectando Python con MongoDB

Insertar un documento

```
colección.insert_one({'aprendizid':3,"nombre":"Maria la  
bandida","Edad":16,"Genero":"F"})
```

Observar en MongoDB Compas

Inserta más de un documento

```
colección.insert_many([{"aprendizid":4,"nombre":"Rosa  
Rico","Edad":36,"Genero":"F"},  
 {"aprendizid":5,"nombre":"Simon Tolomeo","Edad":55,"Genero":"M"}])
```

The screenshot shows the MongoDB Compass interface. On the left, the sidebar lists databases: admin, config, local, and titulada. The titulada database is selected, and its collections are listed: aprendices (which is highlighted in green) and otra. The main pane displays the 'aprendices' collection under the 'Documents' tab. A single document is shown with the following fields:

```
_id: ObjectId('6439a55517d1920eba499621')  
aprendizid: 3  
nombre: "Maria la bandida"  
Edad: 16  
Genero: "F"
```

CONECTANDO MONGODB CON PYTHON



Conectando Python con MongoDB

Mostrar todos los documentos de la colección

```
a=coleccion.find()  
for x in a:  
    print(x["aprendizid"],x["nombre"],x["Genero"],x["Edad"])
```

Mostrar un registro en especial

```
a=coleccion.find({"aprendizid":3})  
for x in a:  
    print(x["aprendizid"],x["nombre"],x["Genero"],x["Edad"])
```

Mostrar todos los documentos ordenados por la edad de forma ascendente

```
a=coleccion.find().sort("Edad",1)  
for x in a:  
    print(x["aprendizid"],x["nombre"],x["Genero"],x["Edad"])
```



Mostrar todos los documentos ordenados por la edad de forma descendente

```
a=coleccion.find().sort("Edad",-1)  
for x in a:  
    print(x["aprendizid"],x["nombre"],x["Genero"],x["Edad"])
```

Cambiar el nombre del aprendiz 3 por “García Rovira”

```
coleccion.update_one({"aprendizid":3},  
{"$set":{"nombre":"García Rovira"}}
```

CONECTANDO MONGODB CON PYTHON



Conectando Python con MongoDB



mongo DB



Cambiar el género por “M” a los que el nombre comience por “Ro”

```
myquery = { "nombre": { "$regex": "^Ro" } }  
newvalues = { "$set": { "Genero": "M" } }
```

```
colección.update_many(myquery,newvalues)
```

Listar solamente el mayor de los aprendices

```
a=colección.find().sort("Edad",-1).limit(1)  
for x in a:  
    print(x["aprendizid"],x["nombre"],x["Genero"],x["Edad"])
```

Borrar el aprendiz cuyo id es igual a 5

```
colección.delete_one({"aprendizid":5})
```

Borrar la colección aprendiz

```
colección.drop()
```

CONECTANDO MONGODB CON PYTHON



INSERT A MONGODB DESDE PYTHON

Importar las librería necesarias

```
import pymongo  
from pymongo import MongoClient  
import pandas as pd
```

Realizar la conexión al servidor

```
cadena="mongodb://localhost:27017"  
cliente=MongoClient(cadena)
```

Cargar la hoja “2,-Categorias” del archivo Excel “Neptuno.xls”

```
x=pd.read_excel("c:/Borrar/Neptuno.xls",sheet_name="2,-Categorias")
```

Cargar los nombres de las columnas

```
k=[]  
for i in x:  
    k.append(i)  
  
z0=x[k[0]]  
z1=x[k[1]]  
z2=x[k[2]]
```

```
i=0  
while i<j:  
    colección.insert_one({"idcategoria":int(z0[i]),"NombreCategoria":z1[i],"Descripcion":z2[i]})  
    i=i+1
```



mongo DB

Cantidad de filas

```
j=len(x)
```

Crear la base de datos “Neptuno.xls”

```
bd=cliente["Neptuno"]
```

Se crea la colección.

```
colección=bd["categorias"]
```

Insertar todas filas de la hoja del libro Excel



CONECTANDO MONGODB CON PYTHON



Verificar en MongoDB si se creo la base de datos “Neptuno”, la colección “categorías”

The screenshot shows the MongoDB Compass interface. On the left, the sidebar lists databases (localhost:27017, 5 DBs) and collections (6 COLLECTIONS). Under 'Neptuno', the 'categorias' collection is selected. The main area displays five documents:

- `_id: ObjectId('645aa74d2da9999c5b1a0045')`
idCategoria: 1
NombreCategoria: "Bebidas"
Descripcion: "Gaseosas, café, té, cervezas y maltas"
- `_id: ObjectId('645aa74d2da9999c5b1a0046')`
idCategoria: 2
NombreCategoria: "Condimentos"
Descripcion: "Salsas dulces y picantes, delicias, comida para untar y aderezos"
- `_id: ObjectId('645aa74d2da9999c5b1a0047')`
idCategoria: 3
NombreCategoria: "Repostería"
Descripcion: "Postres, dulces y pan dulce"
- `_id: ObjectId('645aa74d2da9999c5b1a0048')`
idCategoria: 4
NombreCategoria: "Lácteos"
Descripcion: "Quesos"
- `_id: ObjectId('645aa74d2da9999c5b1a0049')`
idCategoria: 5

Mostrar todos los documentos de la colección “categorías”

```
Docu=colección.find()  
for Muestre in Docu:  
    print(Muestre)
```

The screenshot shows a Jupyter Notebook cell with the following code:

```
In [25]: runfile('C:/Users/Administrador/InsertarMongodb.py', wdir='C:/Users/Administrador')  
{'_id': ObjectId('645aa74d2da9999c5b1a0045'), 'idcategoria': 1, 'NombreCategoria': 'Bebidas', 'Descripcion': 'Gaseosas, café, té, cervezas y maltas'}  
{'_id': ObjectId('645aa74d2da9999c5b1a0046'), 'idcategoria': 2, 'NombreCategoria': 'Condimentos', 'Descripcion': 'Salsas dulces y picantes, delicias, comida para untar y aderezos'}  
{'_id': ObjectId('645aa74d2da9999c5b1a0047'), 'idcategoria': 3, 'NombreCategoria': 'Repostería', 'Descripcion': 'Postres, dulces y pan dulce'}  
{'_id': ObjectId('645aa74d2da9999c5b1a0048'), 'idcategoria': 4, 'NombreCategoria': 'Lácteos', 'Descripcion': 'Quesos'}  
{'_id': ObjectId('645aa74d2da9999c5b1a0049'), 'idcategoria': 5, 'NombreCategoria': 'Granos/Cereales', 'Descripcion': 'Pan, galletas, pasta y cereales'}  
{'_id': ObjectId('645ab5212da9999c5b1a004a'), 'idcategoria': 6, 'NombreCategoria': 'Carnes', 'Descripcion': 'Carnes preparadas'}  
{'_id': ObjectId('645ab5212da9999c5b1a004b'), 'idcategoria': 7, 'NombreCategoria': 'Frutas/Verduras', 'Descripcion': 'Frutas secas y queso de soja'}  
{'_id': ObjectId('645ab5212da9999c5b1a004c'), 'idcategoria': 8, 'NombreCategoria': 'Pescado/Marisco', 'Descripcion': 'Pescados, mariscos y algas'}  
{'_id': ObjectId('645ab5212da9999c5b1a0063'), 'idcategoria': 1, 'NombreCategoria': 'Bebidas', 'Descripcion': 'Gaseosas, café, té, cervezas y maltas'}  
{'_id': ObjectId('645ab5212da9999c5b1a0064'), 'idcategoria': 2, 'NombreCategoria': 'Condimentos', 'Descripcion': 'Salsas dulces y picantes, delicias, comida para untar y aderezos'}  
{'_id': ObjectId('645ab5212da9999c5b1a0065'), 'idcategoria': 3, 'NombreCategoria': 'Repostería', 'Descripcion': 'Postres, dulces y pan dulce'}  
{'_id': ObjectId('645ab5212da9999c5b1a0066'), 'idcategoria': 4, 'NombreCategoria': 'Lácteos', 'Descripcion': 'Quesos'}  
{'_id': ObjectId('645ab5212da9999c5b1a0067'), 'idcategoria': 5, 'NombreCategoria': 'Granos/Cereales', 'Descripcion': 'Pan, galletas, pasta y cereales'}  
{'_id': ObjectId('645ab5212da9999c5b1a0068'), 'idcategoria': 6, 'NombreCategoria': 'Carnes', 'Descripcion': 'Carnes preparadas'}  
{'_id': ObjectId('645ab5212da9999c5b1a0069'), 'idcategoria': 7, 'NombreCategoria': 'Frutas/Verduras', 'Descripcion': 'Frutas secas y queso de soja'}  
{'_id': ObjectId('645ab5212da9999c5b1a006a'), 'idcategoria': 8, 'NombreCategoria': 'Pescado/Marisco', 'Descripcion': 'Pescados, mariscos y algas'}
```



CONECTANDO MONGODB CON PYTHON



Mostrar el “NombreCategoria”

```
Docu=colección.find()  
for Muestre in Docu:  
    print(Muestre['NombreCategoria'])
```

```
In [29]: runfile('C:/Users/Administrador/InsertarMongodb.py', wdir='C:/Users/Administrador')  
Bebidas  
Condimentos  
Repostería  
Lácteos  
Granos/Cereales  
Carnes  
Frutas/Verduras  
Pescado/Marisco
```



mongo DB

CONECTANDO MONGODB CON PYTHON



CONECTANDO POSTGRESQL CON PYTHON

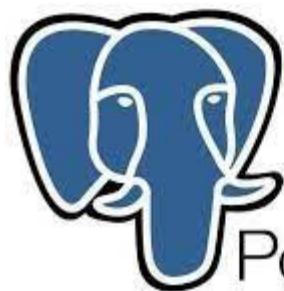
CONECTANDO POSTGRESQL CON PYTHON



PRACTICA CON POSTGRESQL

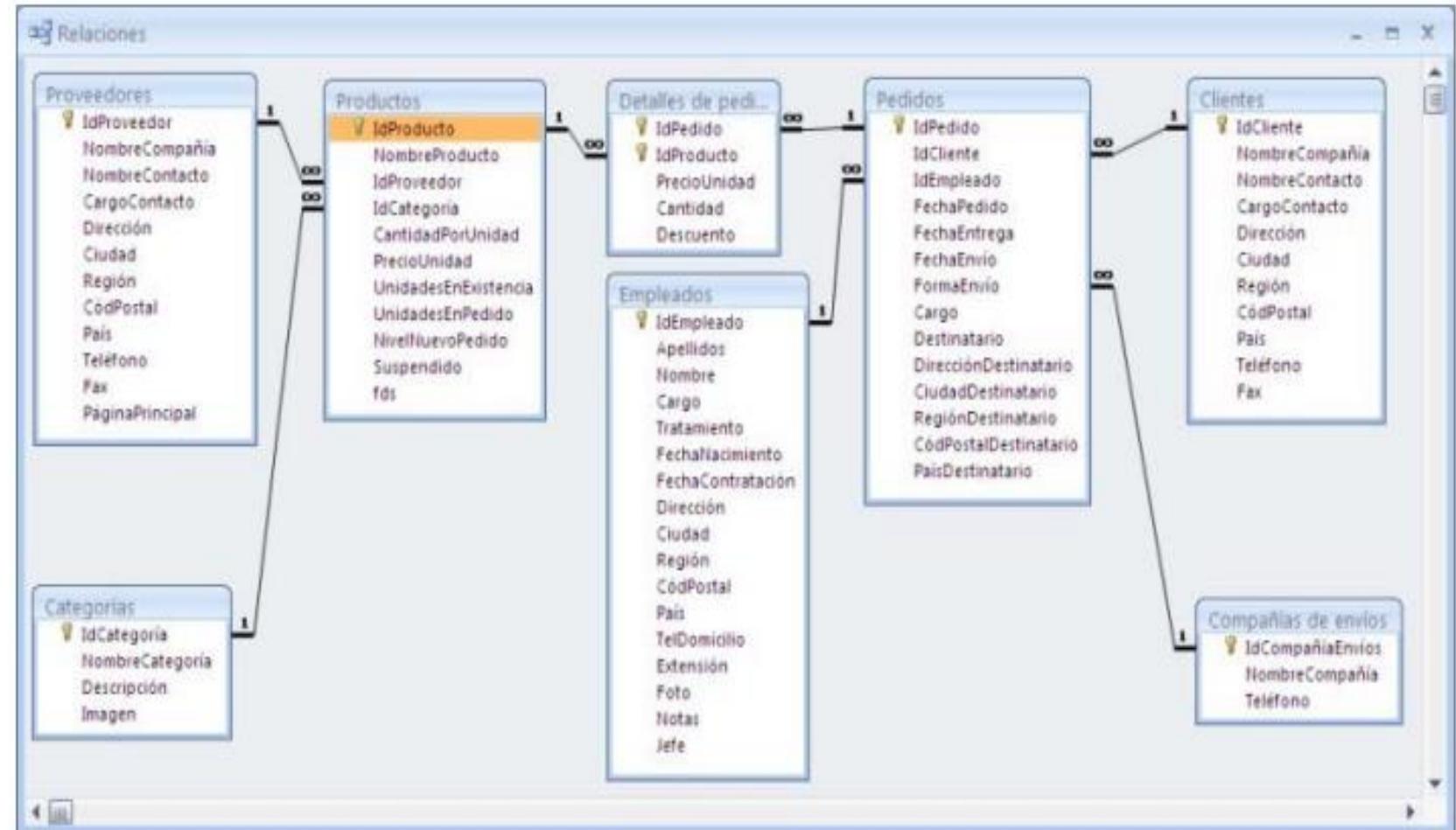
Descargar [PostgreSQL](#) y [Neptuno](#) e instálelo para realizar la práctica con el modelo relacional propuesto.
Importar las librerías necesarias para realizar la práctica

CONECTANDO POSTGRESQL CON PYTHON

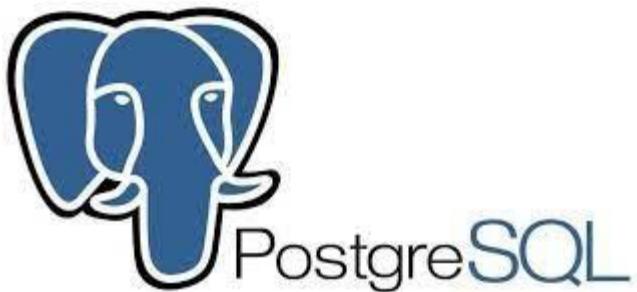


PostgreSQL

NEPTUNO



CONECTANDO POSTGRESQL CON PYTHON



Realizar la conexión al servidor Postgres

```
import psycopg2  
import pandas as pd  
  
conn=psycopg2.connect(  
    host="localhost",  
    user="postgres",  
    password="",  
    database="neptuno"  
)
```

Crear un cursor

```
micursor=conn.cursor()
```

CONECTANDO POSTGRESQL CON PYTHON



Crear la table CATEGORIA en Postgres

```
sq="CREATE TABLE IF NOT EXISTS CATEGORIA(IDCATEGORIA SERIAL PRIMARY  
KEY,NOMBRECATEGORIA TEXT , DESCRIPCION TEXT)"
```

```
micursor.execute(sq)  
conn.commit()
```

Insertar un registro en la tabla CATEGORIA en Postgres

```
sq="insert into categoria(nombrecategoria,descripcion)  
values('Panaderia','Pan quemado')"
```

```
micursor.execute(sq)  
conn.commit()
```

Actualizar un registro de la tabla CATEGORIA

```
sq="update categoria set descripcion='Pan Horneado' where  
idcategoria=1"
```

```
micursor.execute(sq)  
conn.commit()
```

CONECTANDO POSTGRESQL CON PYTHON



Borrar un registro de la tabla CATEGORIA en Postgres

```
sq="delete from categoria where idcategoria=1"
```

```
micursor.execute(sq)  
conn.commit()
```

CONECTANDO POSTGRESQL CON PYTHON

Extraer los datos del libro “Neptuno.xls” de la hoja 5

```
x=pd.read_excel('c:/Borrar/Neptuno.xlsx',sheet_name=5)  
x
```



	IdCategoría	NombreCategoría	Descripción
0	5	Granos/Cereales	Pan, galletas, pasta y cereales
1	7	Frutas/Verduras	Frutas secas y queso de soja
2	2	Condimentos	Salsas dulces y picantes, delicias, comida par...
3	4	Lácteos	Quesos
4	6	Carnes	Carnes preparadas
5	1	Bebidas	Gaseosas, café, té, cervezas y maltas
6	8	Pescado/Marisco	Pescados, mariscos y algas
7	3	reaaaa	Postres, dulces y pan dulce

CONECTANDO POSTGRESQL CON PYTHON



Extraer los datos de las dos columnas en vectores

```
z0=x[ 'NombreCategoría' ]  
z1=x[ 'Descripción' ]
```

**Insertar los datos de las dos columnas en vectores en la tabla
CATEGORIA**

```
i=0  
while i<len(x):  
    sq="insert into categoria(nombrecategoría,descripción)  
values(%s,%s)"  
    sqv=(z0[i],z1[i])  
  
    micursor.execute(sq,sqv)  
    conn.commit()  
    i=i+1
```

CONECTANDO POSTGRESQL CON PYTHON



The screenshot shows the pgAdmin interface. On the left, the Browser pane displays a tree structure of database objects. A red arrow points from the 'categorias' table node in the 'Tables' section to the 'FROM public.categorias' part of the query in the Query Editor. Another red arrow points from the 'Data Output' tab to the results table, which is also highlighted with a red border. The results table contains 8 rows of data.

Query Editor:

```
1 SELECT * FROM public.categorias
2 ORDER BY Idcategoria ASC
```

Data Output:

Idcategoria [PK] integer	nombreCategoria text	descripcion text
1	Condimentos	Salsas dulces y ...
2	Condimentos	Salsas dulces y ...
3	Condimentos	Salsas dulces y ...
4	Condimentos	Salsas dulces y ...
5	Condimentos	Salsas dulces y ...
6	Condimentos	Salsas dulces y ...
7	Condimentos	Salsas dulces y ...
8	Condimentos	Salsas dulces y ...

CONECTANDO POSTGRESQL CON PYTHON

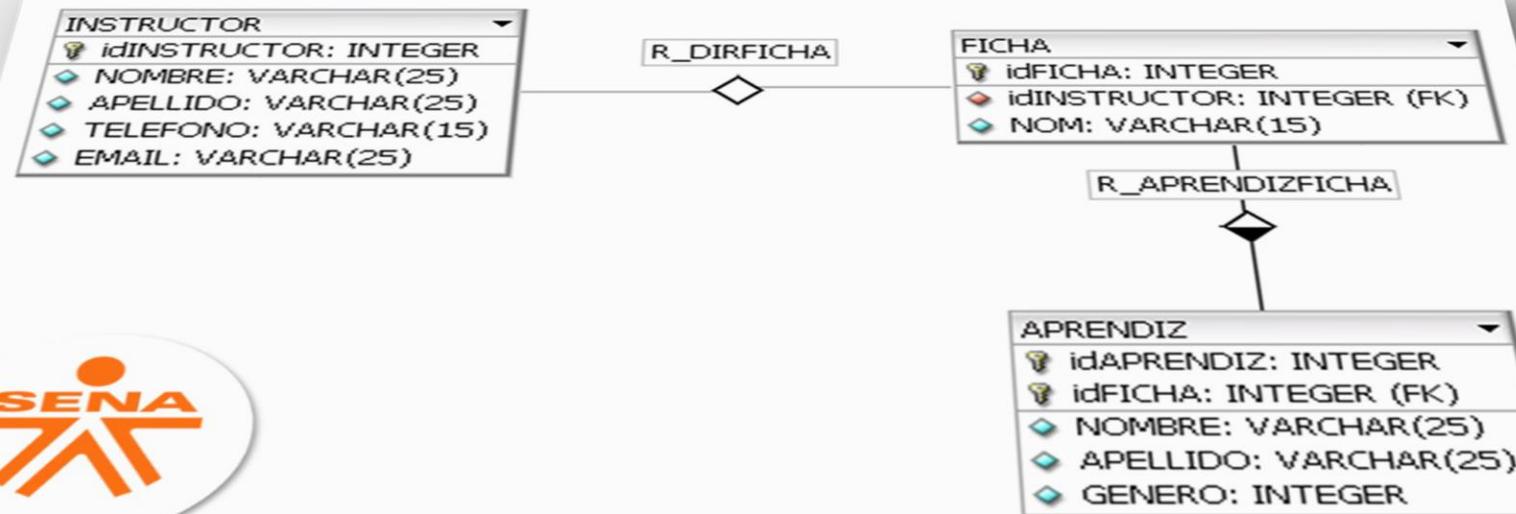


Cerrar el cursor y la conexión

```
micursor.close()
```

```
conn.close()
```

PRACTICA TRABAJO AUTONOMO POSTGRESQL CON PYTHON



PRACTICA POSTGRESQL CON PYTHON



```
CREATE TABLE INSTRUCTOR (IDINSTRUCTOR SERIAL PRIMARY KEY,  
                      NOMBRE VARCHAR(25),  
                      APELLIDO VARCHAR(25),  
                      TELEFONO VARCHAR(15),  
                      EMAIL VARCHAR(25)  
);
```

```
CREATE TABLE FICHA(IDFICHA SERIAL PRIMARY KEY,  
                  IDINSTRUCTOR INTEGER,  
                  NOM VARCHAR(25),  
                  CONSTRAINT INSTRUCTORK FOREIGN KEY(IDINSTRUCTOR)  
                  REFERENCES INSTRUCTOR(IDINSTRUCTOR));
```

```
CREATE TABLE APRENDIZ(IDAPRENDIZ INTEGER,  
                     IDFICHA INTEGER,  
                     NOMBRE VARCHAR(25),  
                     APELLIDO VARCHAR(25),  
                     GENERO INTEGER CHECK(GENERO IN(1,2)),  
                     PRIMARY KEY(IDAPRENDIZ,IDFICHA),  
                     CONSTRAINT FICHAFK FOREIGN KEY(IDFICHA)  
                     REFERENCES FICHA(IDFICHA)  
);
```

Esta práctica se deben crear las tablas (DDL) en pgAdmin de acuerdo al modelo relacional presentado:



PRACTICA POSTGRESQL CON PYTHON



pgAdmin
Management Tools for PostgreSQL

Ingresar los siguientes instructores:

NOMBRE	APELLIDO	TELEFONO	EMAIL
Uldarico	Andrade	5941301	uandrade@senai.edu.co
Erick	Granados	31244455666	egranados@senai.edu.co
Carlos	Patiño	3106665544	cpatino@senai.edu.co

Ingresar las siguientes fichas:

IDINSTRUCTOR	NOMBRE
Erick Granados	2687365
Jose Fegasu	2671743
Carlos Patiño	2465298

Ingresar los siguientes aprendices:

IDAPRENDIZ	IDFICHA	NOMBRE	APELLIDO	GENERO
1010113345	2687365	ELVER	GALINDO	MASCULINO
1011223456	2671743	MARIA	GARCIA	FEMENINO
78456723	2687365	JUANITA	CORZO	FEMENINO

PRACTICA POSTGRESQL CON PYTHON



Instalar las librerías necesarias

```
!pip install psycopg2
```

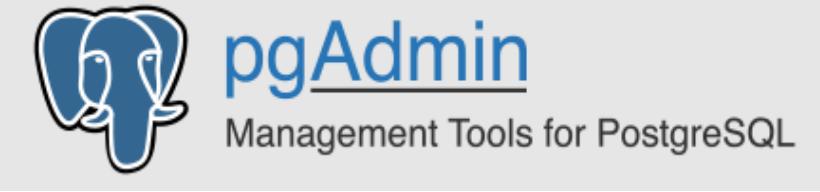
Importar las librerías

```
import psycopg2
```

Realizar la conexión hacia el servidor Postgres

```
cnx=psycopg2.connect(database = "postgres",
                      host="localhost",
                      user="postgres",
                      password="123",
                      port="5432")
```

PRACTICA POSTGRESQL CON PYTHON



Crear la sentencia INSERT de acuerdo a lo solicitado, ejecutar y confirmar la transacción.

```
sql="insert into instructor(nombre,apellido,telefono,email)  
values('Uldarico','Andrade','5941301','uandrade@sena.edu.co')"  
cursor.execute(sql)  
cnx.commit()
```

Nota: Ingresar los siguientes instructores de acuerdo a lo solicitado Se verifica en POSTGRESQL

A screenshot of the pgAdmin Query Editor interface. The title bar shows the connection details: 'public.instructor/postgres/postgres@SenaServer'. The 'Query Editor' tab is active, displaying the following SQL code:

```
1 SELECT * FROM public.instructor  
2 ORDER BY idinstructor ASC
```

The results are displayed in a table titled 'Data Output' with columns: idinstructor, nombre, apellido, telefono, and email. The data shows four rows of instructor information:

	idinstructor [PK] integer	nombre character varying (25)	apellido character varying (25)	telefono character varying (15)	email character varying (25)
1	1	Uldarico	Andrade	5941301	uandrade@sena.edu.co
2	2	Erick	Granados	3124445566	egranados@sena.edu.co
3	3	Carlos	Patiño	3126665544	cpatino@sena.edu.co

PRACTICA TRABAJO AUTONOMO CON PYTHON



- 1. Cargar la hoja de productos del libro Neptuno en una base de datos llamado “Ventas” en la tabla “productos”**
- 2. Lo anterior se debe realizar en Python**
- 3. Se debe repetir esta practica en Mysql, Postgres y sqlite utilizando SQL y ORM**



OBJECT RELATIONAL MAPPING



Object Relational Mapping (ORM)



¿Qué es un ORM?

Es un modelo de programación que permite mapear las estructuras de una base de datos, sobre una estructura lógica de entidades con el objeto de simplificar y acelerar el desarrollo de las aplicaciones.

Ventajas de un ORM

- Facilidad y velocidad de uso
- Abstracción de la base de datos usada.
- Seguridad de la capa de acceso a datos contra ataques.
- Reutilización. Permite utilizar los métodos de un objeto de datos desde distintas zonas de la aplicación, incluso desde aplicaciones distintas.
- Mantenimiento del código. Facilita el mantenimiento del código debido a la correcta ordenación de la capa de datos, haciendo que el mantenimiento del código sea mucho más sencillo.

PeeWee
Python library

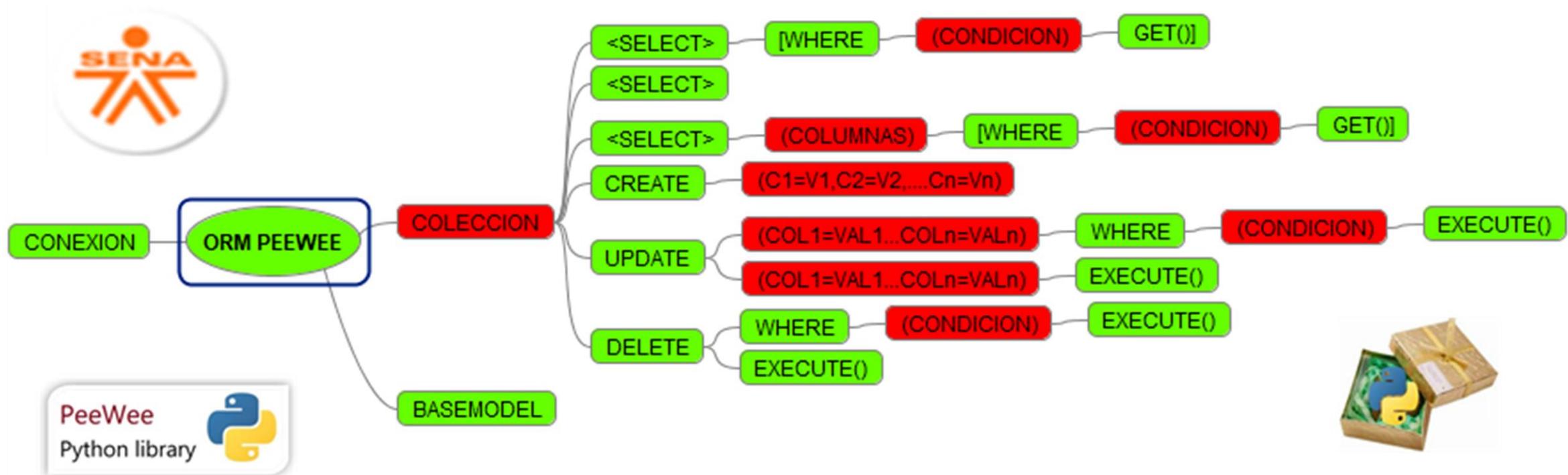


Desventajas de un ORM

- Lentitud en la carga de grandes volúmenes de datos al poner una capa más en el proceso puede mermar el rendimiento; en algunos casos es más rápido utilizar SQL puro.
- La curva de aprendizaje es mas lenta.



Object Relational Mapping (ORM)



Object Relational Mapping (ORM)



Field Type	Sqlite	Postgresql	MySQL
AutoField	integer	serial	integer
BigAutoField	integer	bigserial	bigint
IntegerField	integer	integer	integer
BigIntegerField	integer	bigint	bigint
SmallIntegerField	integer	smallint	smallint
IdentityField	not supported	int identity	not supported
FloatField	real	real	real
DoubleField	real	double precision	double precision
DecimalField	decimal	numeric	numeric
CharField	varchar	varchar	varchar
FixedCharField	char	char	char
TextField	text	text	text
BlobField	blob	bytea	blob
BitField	integer	bigint	bigint
BigBitField	blob	bytea	blob
UUIDField	text	uuid	varchar(40)
BinaryUUIDField	blob	bytea	varbinary(16)
DatetimeField	datetime	timestamp	datetime
DateField	date	date	date
TimeField	time	time	time
TimestampField	integer	integer	integer
IPField	integer	bigint	bigint
BooleanField	integer	boolean	bool
BareField	untyped	not supported	not supported
ForeignKeyField	integer	integer	integer

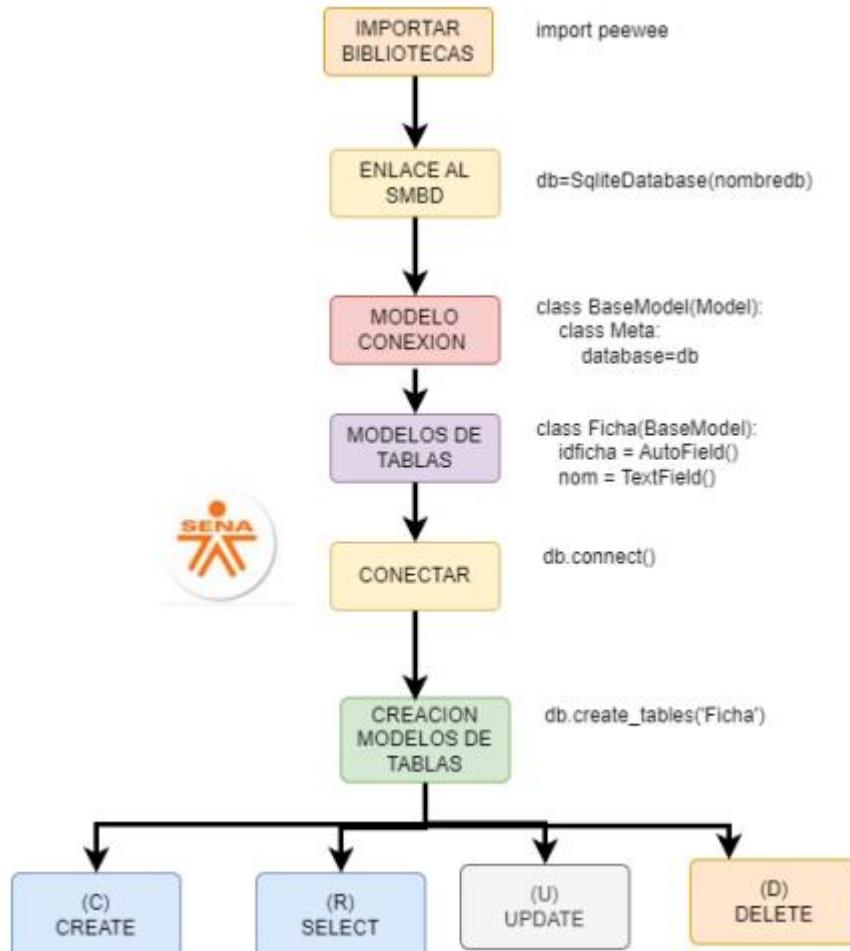
- `null = False` – allow null values
- `index = False` – create an index on this column
- `unique = False` – create a unique index on this column. See also adding composite indexes.
- `column_name = None` – explicitly specify the column name in the database.
- `default = None` – any value or callable to use as a default for uninitialized models
- `primary_key = False` – primary key for the table
- `constraints = None` – One or more constraints, e.g. `[Check('price > 0')]`
- `sequence = None` – sequence name (if backend supports it)
- `collation = None` – collation to use for ordering the field / index
- `unindexed = False` – indicate field on virtual table should be unindexed (**SQLite-only**)
- `choices = None` – optional iterable containing 2-tuples of `value`, `display`
- `help_text = None` – string representing any helpful text for this field
- `verbose_name = None` – string representing the “user-friendly” name of this field
- `index_type = None` – specify a custom index-type, e.g. for Postgres you might specify a `'BRIN'` Or `'GIN'` index.

PeeWee
Python library



<https://docs.peewee-orm.com/en/latest/peewee/models.html>

Object Relational Mapping (ORM)



Instalar el driver ORM

```
#!pip install peewee  
import peewee  
from peewee import *  
import xlrd  
import pandas as pd
```

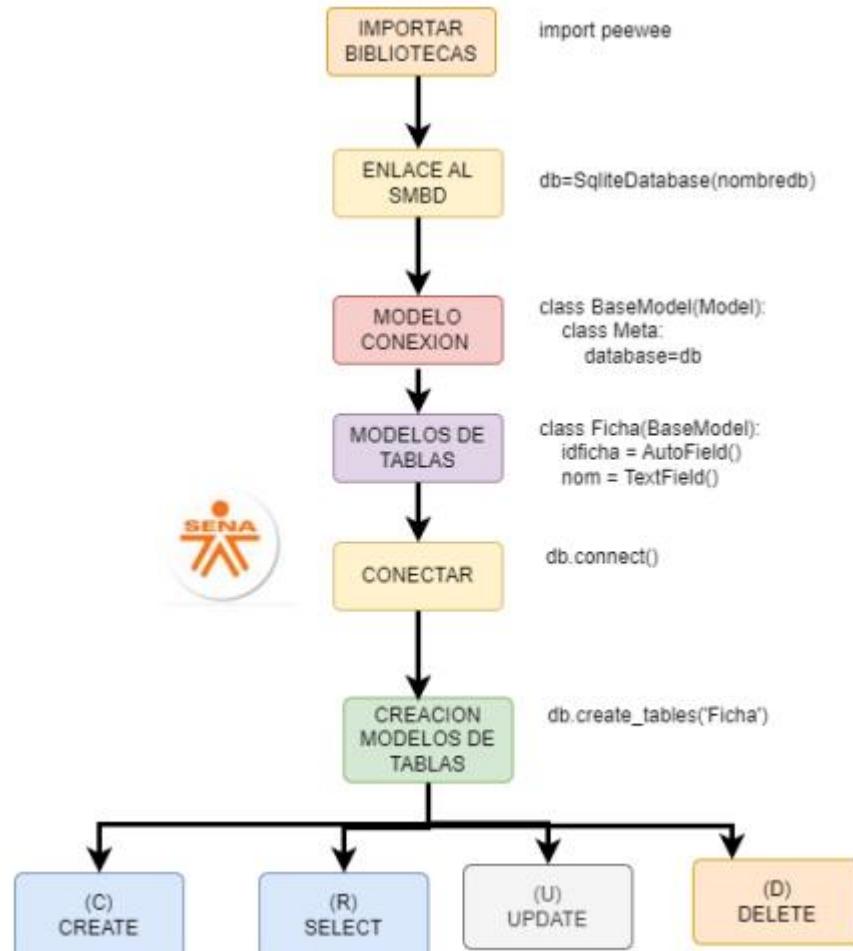
Conectar a una base de datos Postgresql

```
conn = PostgresqlDatabase('neptuno', user='postgres', password='',  
                           host='localhost', port=5432)
```

Crear el modelo de conexión

```
class BaseModel (Model) :  
  
    class Meta:  
        database=conn
```

Object Relational Mapping (ORM)



Mapear la tabla CATEGORIA (modelo de tablas)

```
class categoria(BaseModel):  
    idcategoria=IntegerField(primary_key=True)  
    nombrecategoria=TextField()  
    descripcion=TextField()
```

Conectar a la base de datos

```
conn.connect()
```

Object Relational Mapping (ORM)



Seleccionar todos los registros

```
a=categoria.select()  
for zz in a:  
    print(zz.idcategoria,'|',zz.nombrecategoria,'|',zz.descripcion)
```

```
2 | Granos/Cereales | Pan, galletas, pasta y cereales  
3 | Frutas/Verduras | Frutas secas y queso de soja  
4 | Condimentos | Salsas dulces y picantes, delicias, comida para untar y aderezos  
5 | Lácteos | Quesos  
6 | Carnes | Carnes preparadas  
7 | Bebidas | Gaseosas, café, té, cervezas y maltas  
8 | Pescado/Marisco | Pescados, mariscos y algas  
9 | reaaaa | Postres, dulces y pan dulce
```



Object Relational Mapping (ORM)



Seleccionar un registro

```
a=categoria.select().where(categoria.idcategoria==5).get()  
print(a.nombrecategoría,'|',a.descripcion)
```

Lácteos | Quesos

Crear un registro

```
categoria.create(nombrecategoría='Empanadas',descripcion='Horneadas')
```

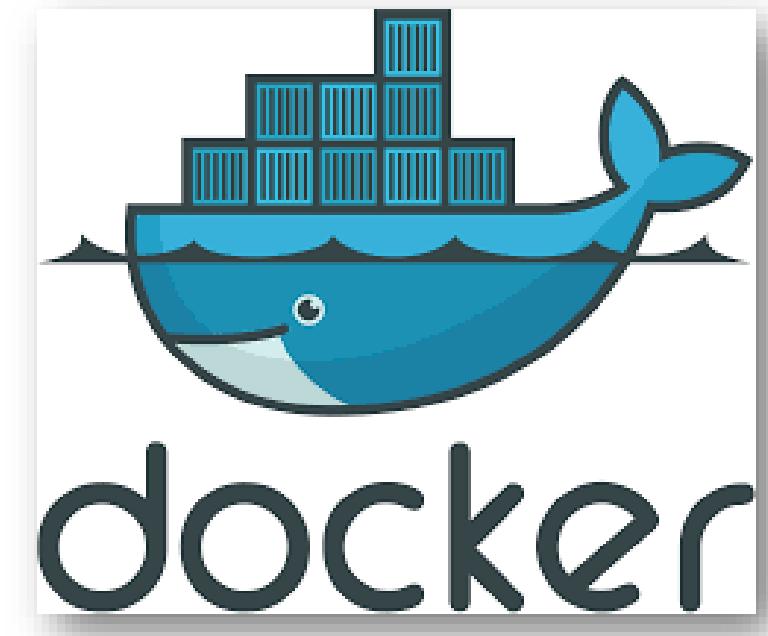
Actualizar un registro

```
categoria.update({"descripcion":  
    "De Lechona"}).where(categoria.idcategoria==10).execute()
```

Borrar un registro

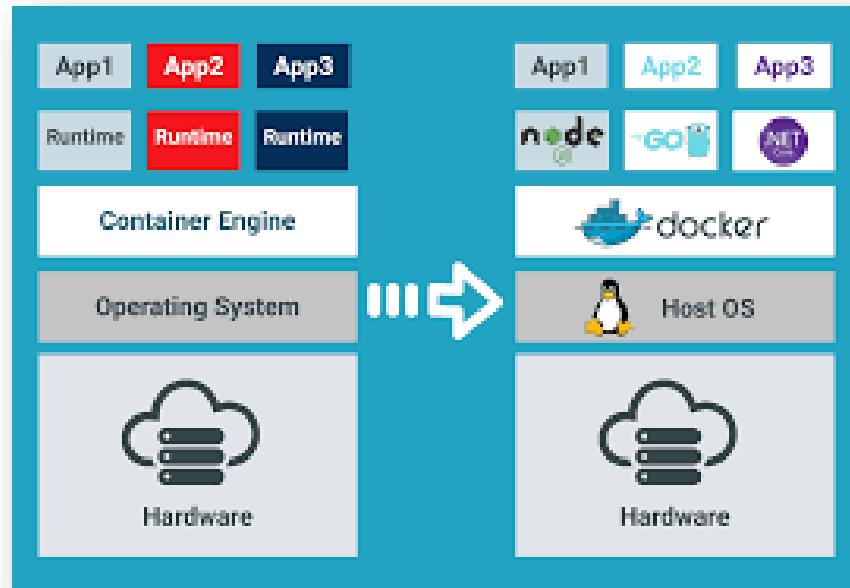
```
categoria.delete().where(categoria.idcategoria==10).execute()
```





¿QUÉ ES DOCKER?

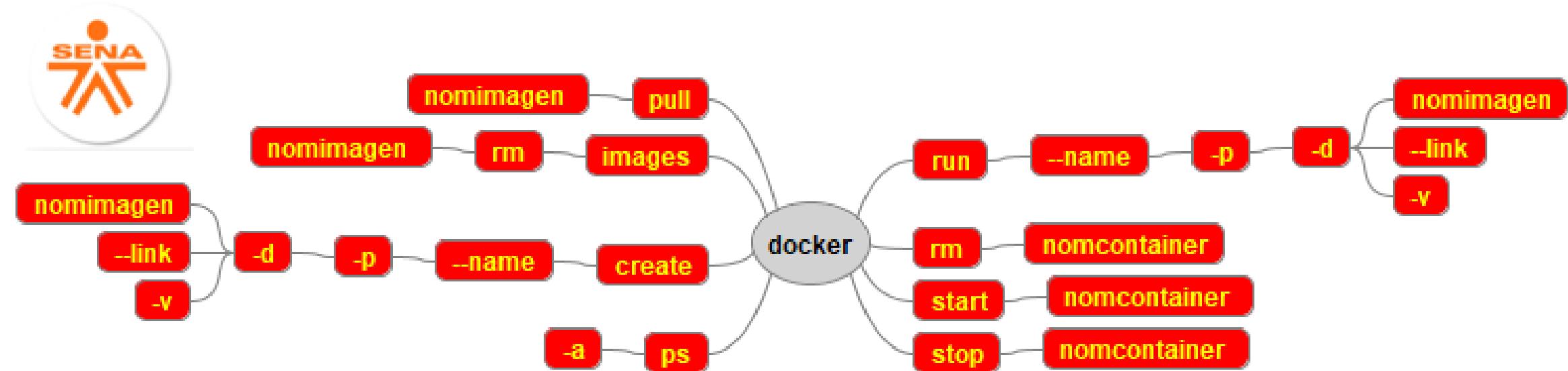
Es un software de código abierto utilizado para desplegar aplicaciones dentro de contenedores virtuales. La contenerización permite que varias aplicaciones funcionen en diferentes entornos complejos. Por ejemplo, Docker permite ejecutar el sistema de gestión de contenidos WordPress en sistemas Windows, Linux y macOS sin ningún problema.



¿QUÉ ES DOCKER?

La principal diferencia con las máquinas virtuales es que los contenedores Docker comparten el sistema operativo del anfitrión, mientras que las máquinas virtuales también tienen un sistema operativo invitado que se ejecuta sobre el sistema anfitrión. Este método de funcionamiento afecta al rendimiento, las necesidades de hardware y la compatibilidad con el sistema operativo.

COMANDOS DOCKER.



Ver video de [DOCKER](#)



PRACTICA CON MYSQL Y DOCKER .

- Instalar [Docker Desktop](#) para poder desarrollar esta práctica.
- Crear el contenedor con la imagen de Mariadb (el comando debe ser en solo una línea)

```
docker run -p 3306:3306 --name mysql -e MYSQL_ROOT_PASSWORD=root  
-d mysql
```

Crear un contenedor para PHP

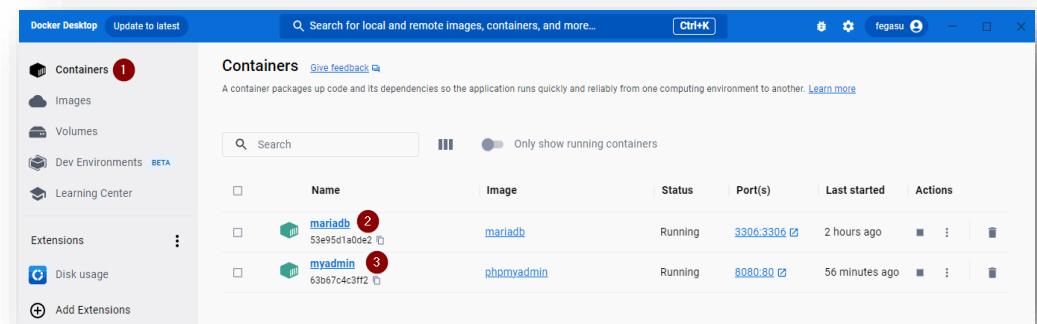
```
docker run -p 9902:80 --name miservidor -v d:/Docker/paginas:/var/www/html -  
-link mysql -d php:8.0-apache
```

Crear el contenedor con la imagen de PhpMyAdmin Mariadb (el comando debe ser en solo una línea)

```
docker run -p 8080:80 --name miadmin -e PMA_HOST=mysql -d  
phpmyadmin/phpmyadmin
```

Se ingresa a la línea de comando bash para iniciar a mysql

```
docker exec -it e7c7e bash
```



A screenshot of the Docker Desktop application interface. On the left, there's a sidebar with icons for Containers (1), Images, Volumes, Dev Environments (BETA), Learning Center, Extensions, and Disk usage. The main area is titled 'Containers' and shows a table of running containers. The table has columns for Name, Image, Status, Port(s), Last started, and Actions. There are three entries:

Name	Image	Status	Port(s)	Last started	Actions
mariadb	mariadb	Running	3306:3306	2 hours ago	[...]
53e95d1a0de2	mariadb	Running	3306:3306	2 hours ago	[...]
myadmin	phpmyadmin	Running	8080:80	56 minutes ago	[...]
63b67c4c3ff2	phpmyadmin	Running	8080:80	56 minutes ago	[...]

DOCKER



Ingresar a mysql

```
root@a906fa108b95:/# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 27
Server version: 10.11.2-MariaDB-1:10.11.2+maria~ubu2204 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use hr
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [hr]> show tables;
+-----+
| Tables_in_hr |
+-----+
| countries   |
| departments |
| emp_details_view |
| employees   |
| job_history |
| jobs         |
| locations   |
| regions     |
+-----+
8 rows in set (0.000 sec)

MariaDB [hr]>
```

Ingresar al PhpMyAdmin desde <http://localhost:8080/>

The screenshot shows the phpMyAdmin interface running on port 8080. The main window displays the 'Configuraciones generales' (General Configuration) panel, which includes options for changing the password, enabling SSL, and selecting character sets. On the left, a sidebar lists databases such as 'hr', 'information_schema', 'localhost', 'mysql', 'performance_schema', 'sys', and 'testdatabase'. The top navigation bar includes tabs for 'Bases de datos', 'SQL', 'Estado actual', 'Cuentas de usuarios', 'Exportar', 'Importar', 'Configuración', 'Replicación', and 'Mas'. The right side of the interface contains several panels: 'Servidor de base de datos' (Server Configuration) listing server details like version and protocol; 'Servidor web' (Web Server) listing software components; and a footer panel for 'phpMyAdmin' with links to version information and documentation.

Instalar JUPYTER

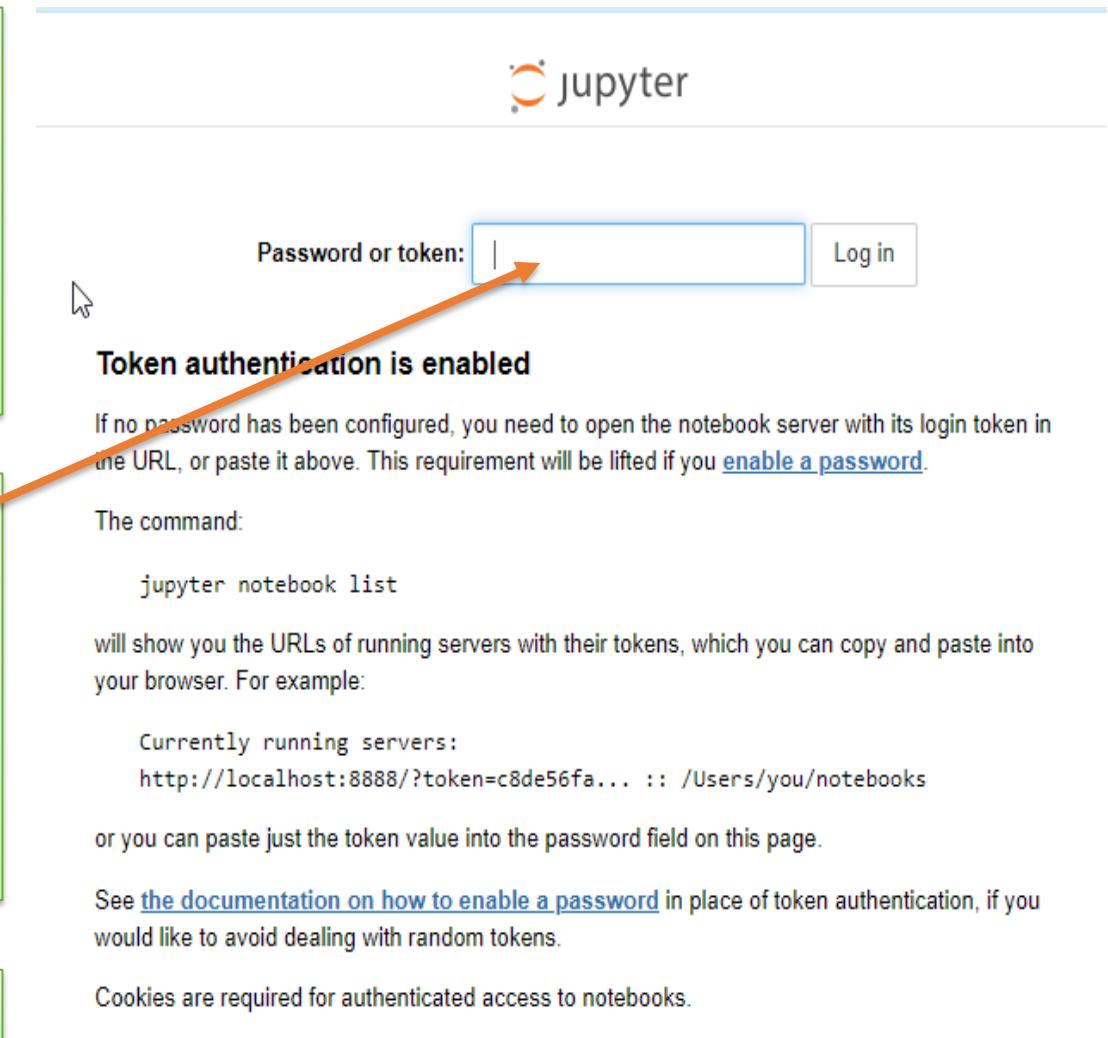
```
docker run -i -t -p 8888:8888  
continuumio/anaconda3 /bin/bash -c " conda  
install jupyter -y --quiet && mkdir -p /opt/notebooks  
&& jupyter notebook --notebook-  
dir=/opt/notebooks --ip='*' --port=8888 --no-browser  
--allow-root"
```

Obtener el token para JUPYTER

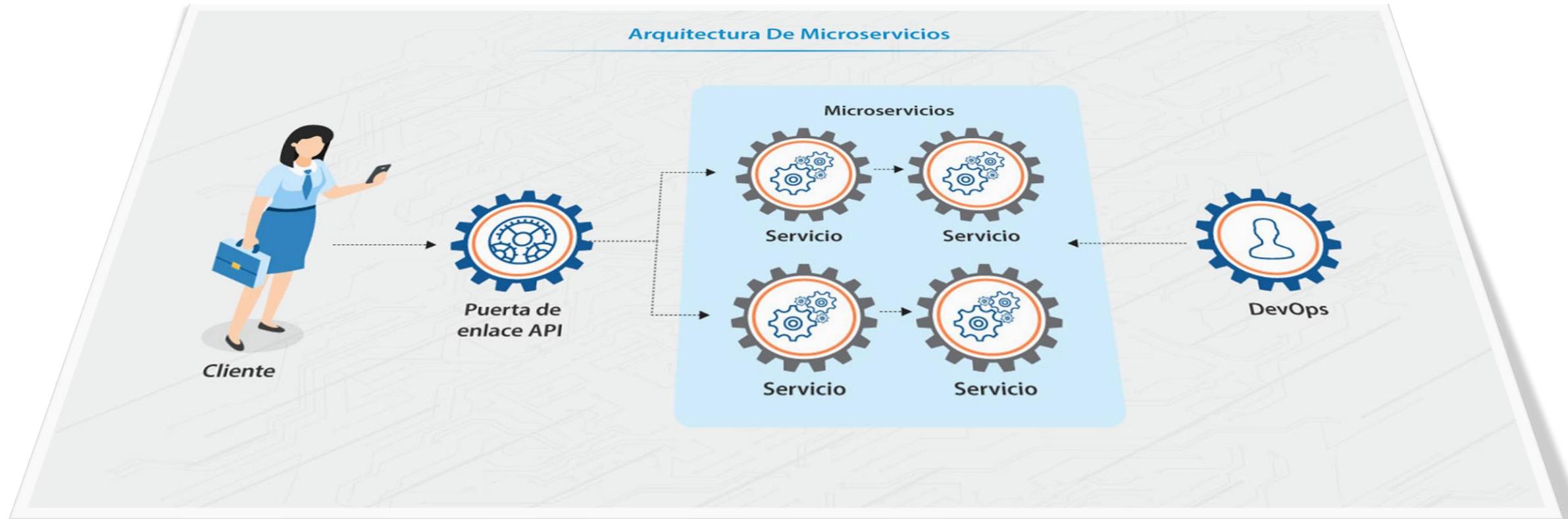
```
C:\Users\Administrador>docker exec -it b60117367bff  
bash  
(base) root@b60117367bff:/# jupyter notebook list  
Currently running servers:  
http://localhost:8888/?token=4f9627f369b0be07c7282fa0  
6aa216d193ec298ea63208f5 :: /opt/notebooks  
(base) root@b60117367bff:/#
```

Abrir JUPYTER

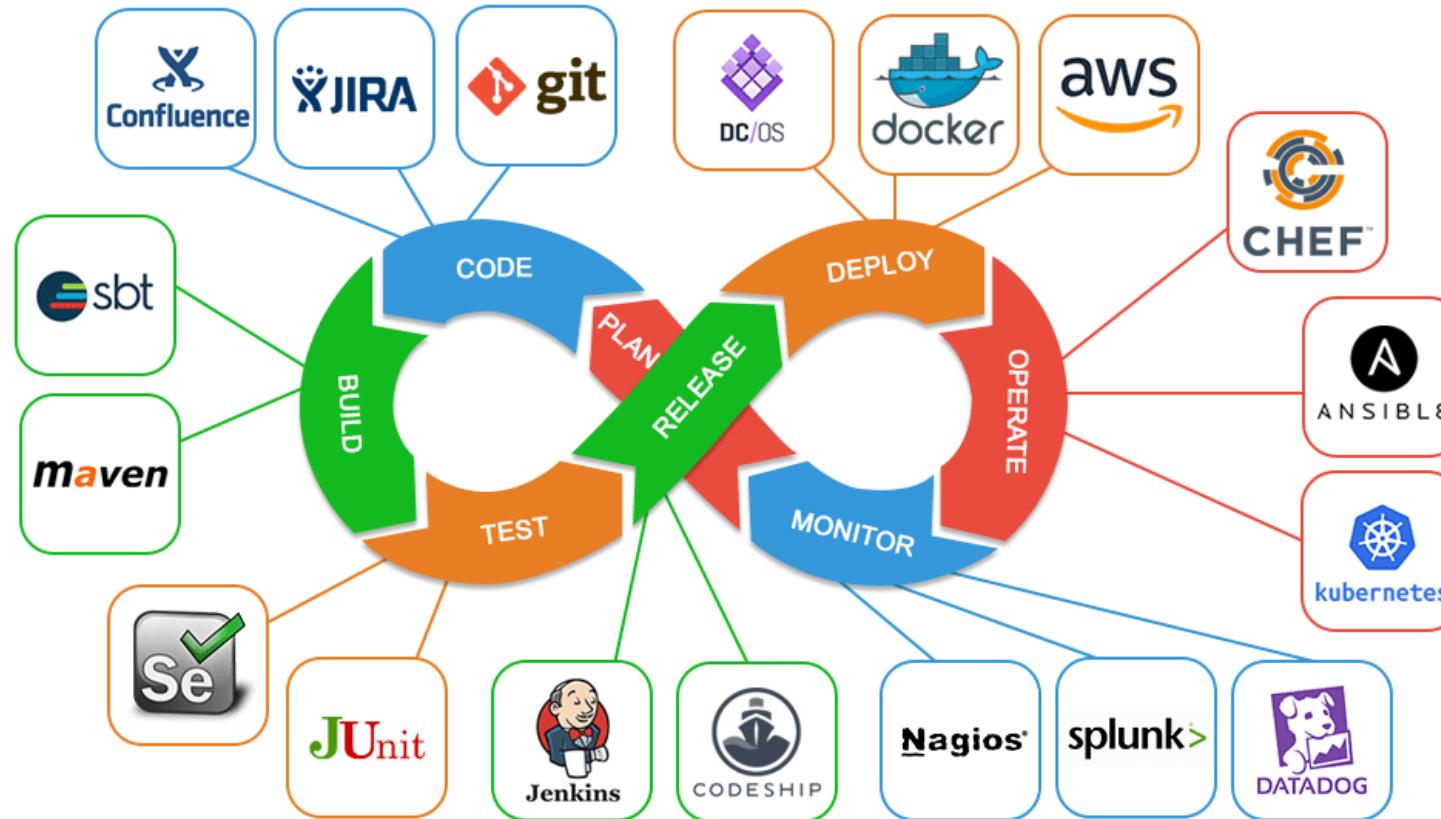
<http://localhost:8888>



MICROSERVICIOS



DESARROLLO Y OPERACIONES



DESARROLLO Y OPERACIONES



CONECTANDO CON API

INTRODUCCIÓN.

Las API o interfaz de programación de aplicaciones, son un conjunto de funciones que se ofrecen mediante bibliotecas de programación para su uso.

Para entender el concepto de API debemos leer los siguientes términos:



Interfaz: capa de extracción para que las aplicaciones se comuniquen.

- **API:** del inglés Application Programming Interface, que permite compartir datos y procesos entre ellos.
- **Arquitectura de software:** Es la forma como está diseñado un sistema, como se comunican al interior, como están organizados sus componentes.
- **Servicio WEB:** Permite una comunicación entre aplicaciones de una red usando el protocolo HTTP.
- **REST:** Es una arquitectura, del inglés Representational State Transfer o Representación de transferencia de estado, permitiendo que datos puede acceder, revisar o manipular otra aplicación.
- **XML:** Formato tradicional para enviar datos, del inglés Extensible Markup Language.
- **JSON:** Formato actual para el intercambio de datos, del inglés JavaScript Object Notation
- **TOKEN: API** pueden ser públicas o privadas, esta última requiere de una autenticación que al hacerlo te devuelve un token que es un objeto que contiene datos de la autenticación y su formato es JWT

CONECTANDO CON API

Tipos de API

- **Locales:** Se ejecutan dentro del mismo entorno. Por ejemplo cuando se desarrolla en Android y se quiere que el celular vibre, entonces debo utilizar la API del smartphone.
- **Remotas:** Cuando se consumen datos que están en otro lugar o **Servicios WEB**
 - **SOAP** (Simple Object Access Protocol)
 - **REST** (Representational state transfer) y al utilizarlo lo llamamos RESTFUL



CONECTANDO CON API

Cada recurso que se consuma tiene un identificador único (URI); cuando se consulta un recurso el servidor puede contestar con los siguientes códigos:

- 2XX: Todo fue exitoso
- 3XX: Significan redirecciones
- 4XX: Solicitudes invalidas
- 5XX: Errores directamente en el servidor

[Fuente](#)



CONECTANDO CON API

MÉTODOS HTTP

- **GET:** Este método se emplea para leer una representación de un resource. En caso de respuesta positiva (200 OK), devuelve la representación en un formato concreto: HTML, XML, JSON o imágenes, JavaScript, CSS, etc. [Leer mas](#)

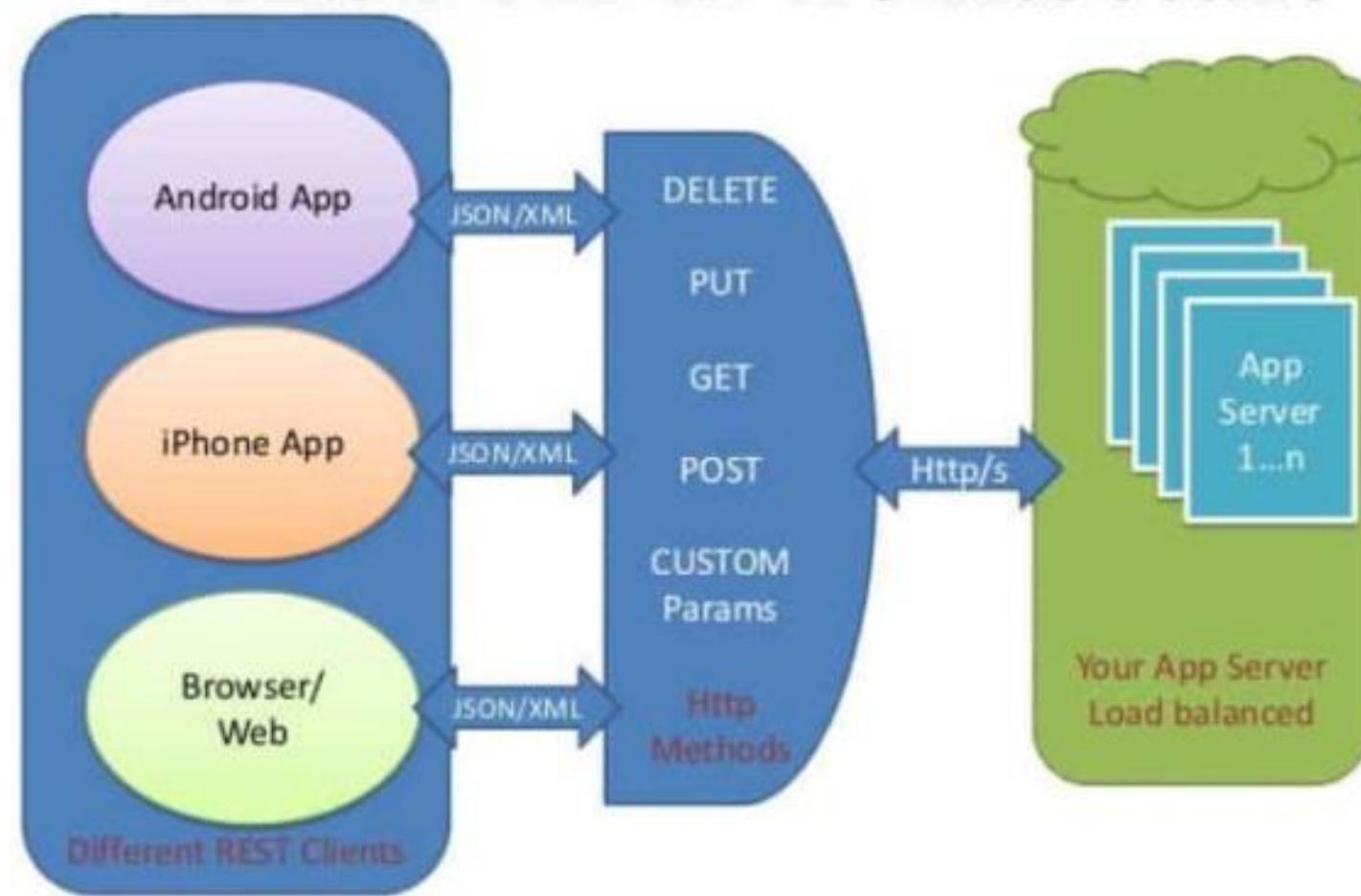
- **POST:** Se utiliza POST por las limitaciones de GET. En caso de respuesta positiva devuelve 201 (*created*). Los POST requests se envían normalmente con formularios [Leer mas..](#)



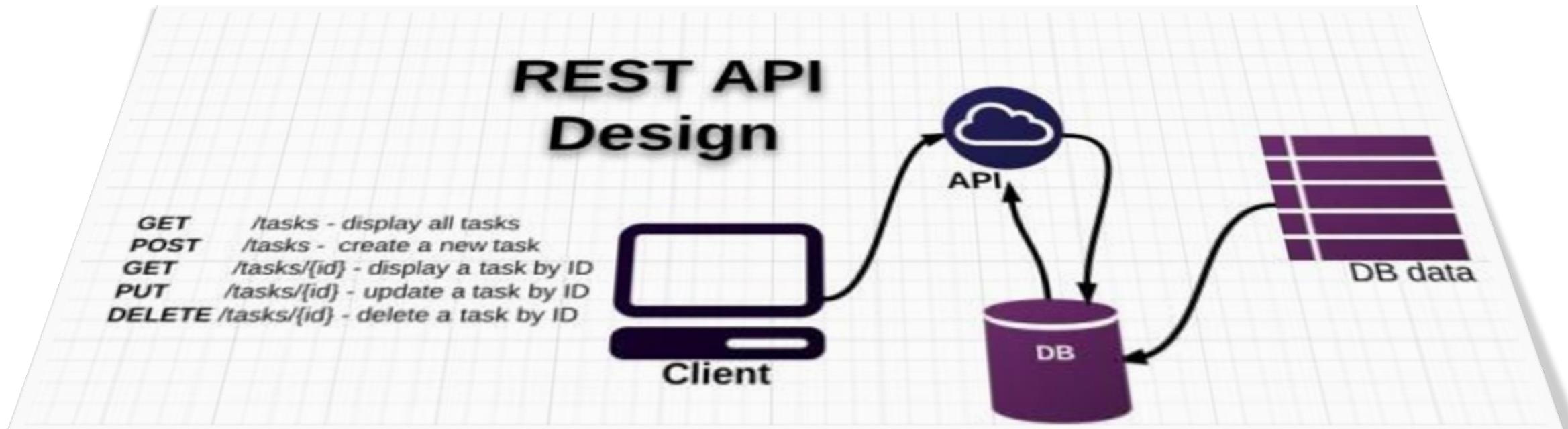
- **PUT:** Utilizado normalmente para actualizar contenidos. Tampoco muestra ninguna información en la URL. En caso de éxito devuelve 201 (*created*, en caso de que la acción haya creado un elemento) o 204 (*no response*, si el servidor no devuelve ningún contenido). A diferencia de POST es idempotente, (*propiedad para realizar una acción varias veces y aun así conseguir el mismo resultado que se obtendría si se hicieran una sola vez*), si se crea o edita un resource con PUT y se hace el mismo request otra vez, el resource todavía está ahí y mantiene el mismo estado que en la primera llamada. Si con una llamada PUT se cambia aunque sea sólo un contador en el resource, la llamada ya no es idempotente, ya que se cambian contenidos. [Leer mas...](#)

- **DELETE:** Simplemente elimina un resource identificado en la URI. Si se elimina correctamente devuelve 200 junto con un body response, o 204 sin body. DELETE, al igual que PUT y GET, también es idempotente. [Leer más...](#)

REST API Architecture



CONECTANDO CON API



PRACTICA CONECTANDO CON API DESDE PYTHON



Crear en MYSQL la tabla USUARIO e Insertar los siguientes valores:



```
CREATE DATABASE APIDB;  
USE APIDB
```

```
CREATE TABLE IF NOT EXISTS USUARIO (  
idUsuario INTEGER PRIMARY KEY AUTO_INCREMENT,  
login varchar ( 30 ),  
nombre TEXT,  
apellido TEXT,  
email TEXT  
);
```

```
INSERT INTO USUARIO (idUsuario,login,nombre,apellido,email) VALUES (2,'fegasu','JOSE FERNANDO','GALINDO SUAREZ',  
'fegasu@misena.edu.co');  
INSERT INTO USUARIO (idUsuario,login,nombre,apellido,email) VALUES (38,NULL,'Carmen Julia','Vargas','cvargas@gmail.com');
```

CONSTRUIR UNA API CON RESTFULL

```
from flask import Flask,render_template,request  
import json  
import mysql.connector  
import pandas as pd
```

1

```
conn=mysql.connector.connect(  
    host="localhost",  
    user="root",  
    passwd="",  
    database="APIDB",  
    port=3306  
)  
cursor=conn.cursor()
```

2

```
def Ejecutar(sql):  
    cursor=cnn.cursor()  
    cursor.execute(sql)  
    cnn.commit()  
    return json.dumps("200")
```

3

PRACTICA DEL TALLER
CREANDO LOS MÉTODO DE CONSULTAS DE BASE DE DATOS

```
if __name__ == "__main__":  
    app.run(debug=True)
```

5

```
def EjecutarQuery(sql):  
    cursor.execute(sql)  
    row=cursor.fetchall()  
    return json.dumps(row)  
app=Flask(__name__)
```

4

API REST

Crear Usuario	POST	/usuario/i
Obtener Usuarios	GET	/usuario/
Obtener un Usuario	GET	/usuario/s/{id}
Actualizar un Usuario	PUT	/usuario/u/{id}
Eliminar un Usuario	DELETE	/usuario/d/{id}

CONSTRUIR UNA API CON RESTFULL



CREANDO LA RUTA PARA CONSULTAR CON EL MÉTODO GET

Obtener usuarios GET /usuario

Obtener un usuario GET /usuario/s/{id}

```
@app.route("/usua",methods=[ 'GET' ])
def get_usuarios():
    sql="select * from usuario order by 1"
    listar=EjecutarQuery(sql)
    return listar

@app.route("/usua/s/<id>",methods=[ 'GET' ])
def get_usuarios1(id):
    sql="select * from usuario where idusuario="+str(id)+" order by 1"
    listar=EjecutarQuery(sql)
    return listar
```

CONSTRUIR UNA API CON RESTFULL



http://127.0.0.1:5000/usua

```
[[2, "fegasu", "JOSE FERNANDO", "GALINDO SUAREZ", "fegasu@misena.edu.co"], [38, null, "Carmen Julia", "Vargas", "cvargas@gmail.com"], [42, "cgomez", "carlos", "Rodriguez", "cgomez@gmail.com"]]
```

http://127.0.0.1:5000/usua/s/38

```
[[38, null, "Carmen Julia", "Vargas", "cvargas@gmail.com"]]
```

http://127.0.0.1:5000/usua

GET http://127.0.0.1:5000/usua

HEADERS AUTHORISATION 0 ACTIONS 0 CONFIG CODE SNIPPETS

COPY Text editor

Add a header to the HTTP request.

+ ADD

Response

200 OK Time: 10 ms Size: 192 Bytes

```
1 | [[2, "fegasu", "JOSE FERNANDO", "GALINDO SUAREZ", "fegasu@misena.edu.co"], [38, null, "Carmen Julia", "Vargas", "cvargas@gmail.com"], [42, "cgomez", "carlos", "Rodriguez", "cgomez@gmail.com"]]
```

... 127.0.0.1:5000/usua/s/38

GET http://127.0.0.1:5000/usua/s/38

HEADERS AUTHORISATION 0 ACTIONS 0 CONFIG CODE SNIPPETS

COPY Text editor

Add a header to the HTTP request.

+ ADD

Response

200 OK Time: 8 ms Size: 59 Bytes

```
1 | [[38, null, "Carmen Julia", "Vargas", "cvargas@gmail.com"]]
```

EJECUTANDO EL SERVICIO POST QUE INSERTA UN REGISTRO EN LA BASE DE DATOS

Insertar un usuario POST /usuario/i

```
@app.route("/usua/i",methods=[ 'POST'])
def post_iusuario():
    json=request.get_json(force=True)
    sql="insert into usuario(login,nombre,apellido,email)
values(%s,%s,%s,%s)"
    sqlv=(json["login"],json["nombre"],json["apellido"],json["email"])
    cursor.execute(sql,sqlv)
    conn.commit()
    return "200"
```

CONSTRUIR UNA API CON RESTFULL



http://127.0.0.1:5000/usua/i +

POST ▼ http://127.0.0.1:5000/usua/i ▶ ⋮

HEADERS BODY AUTHORIZATION 0 ACTIONS 0 CONFIG CODE SNIPPETS

Raw input ▼

```
1 {  
2   "login": "mtorres",  
3   "nombre": "Maria Teresa",  
4   "apellido": "Torres",  
5   "email": "mtorres@gmail.com"  
6 }
```

Mime type ▼

Response X CLEAR

200 OK Time: 437 ms Size: 3 Bytes ⋮

http://127.0.0.1:5000/usua +

GET ▼ http://127.0.0.1:5000/usua ▶ ⋮

HEADERS AUTHORIZATION 0 ACTIONS 0 CONFIG CODE SNIPPETS

COPY Text editor

Add a header to the HTTP request.

+ ADD

Response X CLEAR

200 OK Time: 11 ms Size: 311 Bytes ⋮

```
1 [[2, "fegasu", "JOSE FERNANDO", "GALINDO SUAREZ", "fegasu@misena.edu.co"], [38, null, "Carmen Julia", "Vargas", "cvargas@gmail.com"], [42, "cgomez", "carlos", "Rodriguez", "cgomez@gmail.com"], [44, "ez", "carlos", "Rodriguez", "cgomez@gmail.com"], [45, "mtorres", "Maria Teresa", "Torres", "matorres@gmail.com"]]
```

EJECUTANDO EL SERVICIO PUT QUE ACTUALIZA UN REGISTRO EN LA BASE DE DATOS

Actualizar usuario PUT /usuario/u/{id}

```
@app.route("/usua/u/<id>",methods=['PUT'])
def post_ausuario(id):
    json=request.get_json(force=True)
    sql="update usuario set login=%s,nombre=%s,apellido=%s,email=%s where
idusuario="+str(id)
    sqlv=(json["login"],json["nombre"],json["apellido"],json["email"])
    cursor.execute(sql,sqlv)
    conn.commit()
    return "200"
```

CONSTRUIR UNA API CON RESTFULL



...127.0.0.1:5000/usua/u/44x +

PUT ▼ http://127.0.0.1:5000/usua/u/44

▼ ▼ ▼ ▼ ▼

HEADERS BODY AUTHORIZATION 0 ACTIONS 0 CONFIG CODE SNIPPETS

Raw input ▼

```
1 {  
2   "login": "gbeltran",  
3   "nombre": "Gustavo",  
4   "apellido": "Beltran",  
5   "email": "gbeltran@gmail.com"  
6 }
```

Mime type ▼

Response X

▼ ▼ ▼ ▼ ▼

200 OK CLEAR

Time: 11 ms Size: 3 Bytes ▼

http://127.0.0.1:5000/usua x +

GET ▼ http://127.0.0.1:5000/usua

▼ ▼ ▼ ▼ ▼

HEADERS AUTHORIZATION 0 ACTIONS 0 CONFIG CODE SNIPPETS

COPY Text editor

Add a header to the HTTP request.

+ ADD

Response X

▼ ▼ ▼ ▼ ▼

1 [[2, "fegasu", "JOSE FERNANDO", "GALINDO SUAREZ", "fegasu@misenas.edu.co"], [38, null, "Carmen Julia", "Vargas", "cvargas@gmail.com"], [42, "cgomez", "carlos", "Rodríguez", "cgomez@gmail.com"], [44, "gbeltran", "Gustavo", "Beltran", "gbeltran@gmail.com"], [45, "mtorres", "Maria Teresa", "Torres", "mtorres@gmail.com"]]

EJECUTANDO EL SERVICIO DELETE QUE ELIMINA UN REGISTRO EN LA BASE DE DATOS

Borrar un usuario **DELETE** /usuario/d/{id}

```
@app.route("/usua/d/<id>",methods=[ 'DELETE'])
def post_dusuario(id):
    sql="delete from usuario where idusuario="+str(id)
    cursor.execute(sql)
    conn.commit()
    return "200"
```

CONSTRUIR UNA API CON RESTFULL



... 127.0.0.1:5000/usua/d/38 × +

DELETE ▾ http://127.0.0.1:5000/usua/d/38 edit ▶ ⋮

HEADERS BODY AUTHORIZATION 0 ACTIONS 0 CONFIG CODE SNIPPETS

COPY Text editor

Add a header to the HTTP request.

⊕ ADD

⋮ Response × CLEAR

200 OK Time: 579 ms Size: 3 Bytes ⋮

CONECTANDO CON API



CREAR UN MÉTODO PARA EJECUTAR POR TIPO DE SERVICIO

```
import urllib.request
import json
import requests
from requests import Session

def EjecutaRest(url,metodo,datos):
    url=url.replace(chr(32),'+')
    miurl=None
    try:
        if metodo=="POST":
            miurl=requests.post(url,json=datos)
        if metodo=="PUT":
            miurl=requests.put(url,json=datos)
        if metodo=="DELETE":
            miurl=requests.delete(url)
    return miurl
except ValueError:
    messagebox.showerror("Rutinas/EjecutaRest","Ocurrio un error")
```

CONSUMIR LOS SERVICIOS POST DE LA API

```
datos={  
    "login":"mlb",  
    "nombre":"Maria",  
    "apellido":"La bandida",  
    "email":"emadera@gmail.com"  
}
```

```
print(EjecutaRest('http://127.0.0.1:5000/usuario','POST',datos))
```

CONSUMIR LOS SERVICIOS PUT DE LA API

```
datos={  
    "login":"cgr",  
    "nombre":"Celia",  
    "apellido":"Gacha Rico",  
    "email":" cgr @gmail.com"  
}
```

```
print(EjecutaRest('http://127.0.0.1:5000/usua/u/3','PUT',datos))
```

CONSUMIR LOS SERVICIOS DELETE DE LA API

```
datos={  
    "login": "cgr",  
    "nombre": "Celia",  
    "apellido": "Gacha Rico",  
    "email": " cgr @gmail.com"  
}
```

```
print(EjecutaRest('http://127.0.0.1:5000/usua/d/3','DELETE',datos))
```

EJEMPLO CONSUMIR LOS SERVICIOS GET DE LA API

```
import urllib.request
import json
def CargaDatos(url):
    url=url.replace(chr(32),' ')
    miurl=urllib.request.urlopen(url)
    miarreglo=json.loads(miurl.read().strip())
    miarreglo.reverse()
    return miarreglo
Datos=CargaDatos('http://127.0.0.1:5000/usua')
print(Datos)
```

[[6, 'mlb', 'Celia', 'Gacha Rico', 'cgachar@gmail.com'], [5, 'mlb', 'Celia', 'Gacha Rico', 'cgachar@gmail.com'], [2, 'mlb', 'Maria', 'La bandida', 'emadera@gmail.com']]

CONECTANDO CON API



EJEMPLO CONSUMIR LOS SERVICIOS GET DE LA API

Crear un archivo llamado rutinas.py

```
import mysql.connector
import urllib.request
import json
from tkinter import *
from tkinter import ttk

def CargaDatos(url):
    url=url.replace(chr(32),' ')
    miurl=urllib.request.urlopen(url)
    miarreglo=json.loads(miurl.read().strip())
    miarreglo.reverse()
    return miarreglo
```

```
class kinter:
    def Ventana(self,titu,tam):
        v=Tk();v.title(titu);v.resizable(0,0);v.geometry(tam)
        return v
    def AgregarLista(self,Datos,lista):
        for i in range(len(Datos)):
            lista.insert(Datos[i][0],Datos[i][2]+
"+Datos[i][3]")
            lista.pack()
    def CrearLista(self,v):
        nom=Listbox(v);nom.grid(column=0,row=0)
        return nom
```



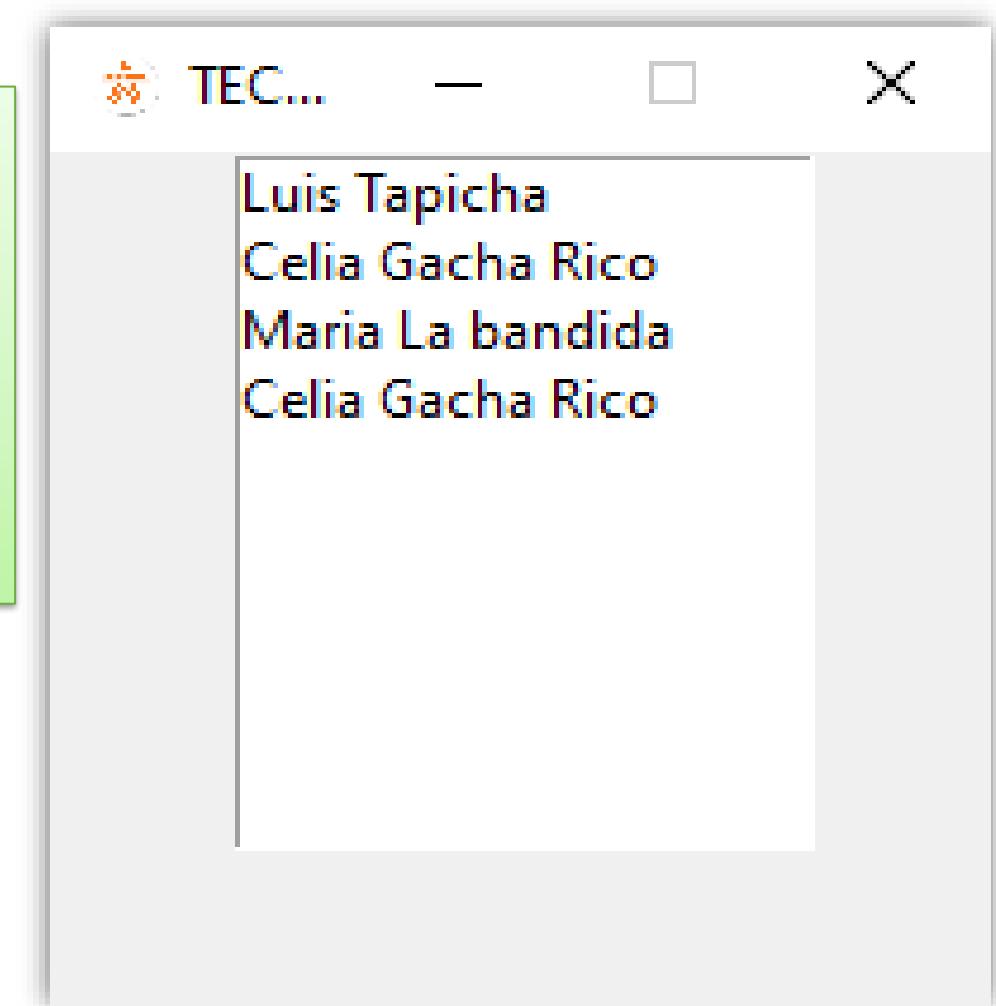
Tkinter The Python logo, which is a stylized yellow and blue 'P' shape.

EJEMPLO CONSUMIR LOS SERVICIOS GET DE LA API

Crear un archivo llamado consultaapi.py

```
import pandas as pd
from rutinas import *

Datos=CargaDatos('http://127.0.0.1:5000/usua')
Datos1=pd.DataFrame(CargaDatos('http://127.0.0.1:5000/usua/c/usuario'))
p1=Tkinter()
v0=p1.Ventana('TECNOLOGIAS EMERGENTES CON PYTHON V1.0 2023','200x200')
v0.iconbitmap("img/logosenacir.ico")
lista2=p1.CrearLista(v0)
p1.AgregarLista(Datos,lista2)
v0.mainloop()
```



Tkinter 

RETO EVIDENCIA



1. Realizar un video en YouTube que muestre la creación de una base de datos aplicando operaciones con SQL y ORM desde Python, con un DATASET de los datos personales de sus compañeros de ficha (nombres, apellidos, dni, tipo dni, genero, fecha de nacimiento, ciudad de nacimiento), en MySQL, POSTGRESQL y MongoDB.
2. Se debe realizar desde Python.
3. Enviar el script de creación del punto 2.

Nota: Esta evidencia se debe realizar en grupo de 4 aprendizaje, se debe escoger un motor de base de datos (MySQL, POSTGRESQL y MongoDB), y se debe realizar todas las operaciones aprendidas en Python y registrado en el video. Se debe subir individualmente indicando los nombres de sus compañeros de equipo.

CONECTIVIDAD A BASE DE DATOS



Atribución, no comercial,compartir igual

Este material puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de licencia que el trabajo original.



Creative Commons



G R A C I A S

Línea de atención al ciudadano: 01 8000 910270
Línea de atención al empresario: 01 8000 910682



@SENACOMUNICA

www.sena.edu.co