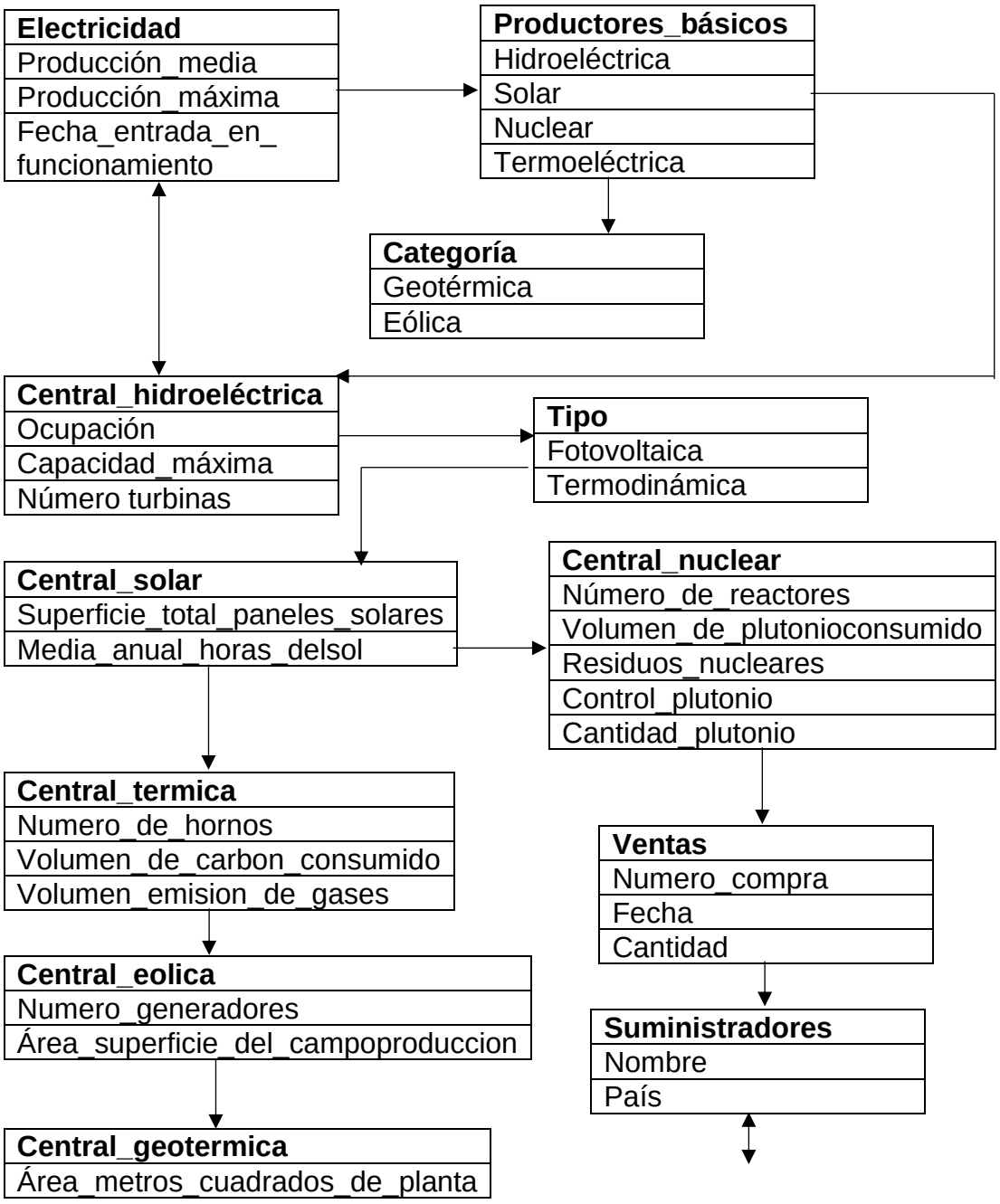
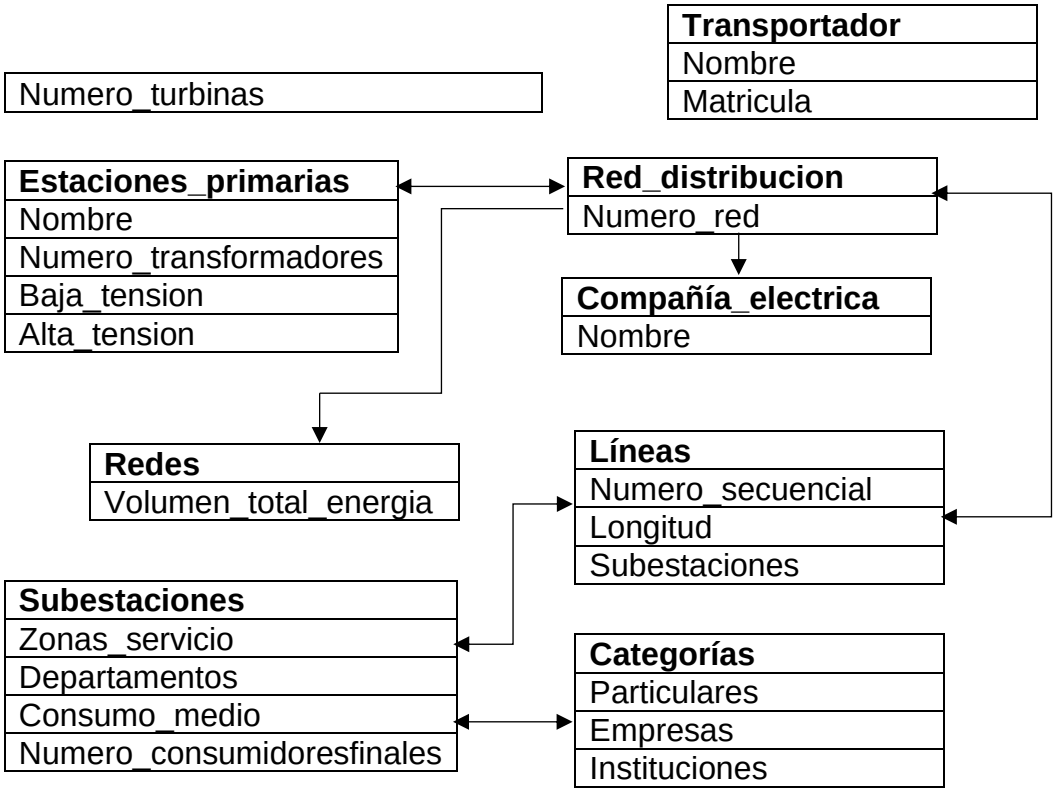
1. DBA

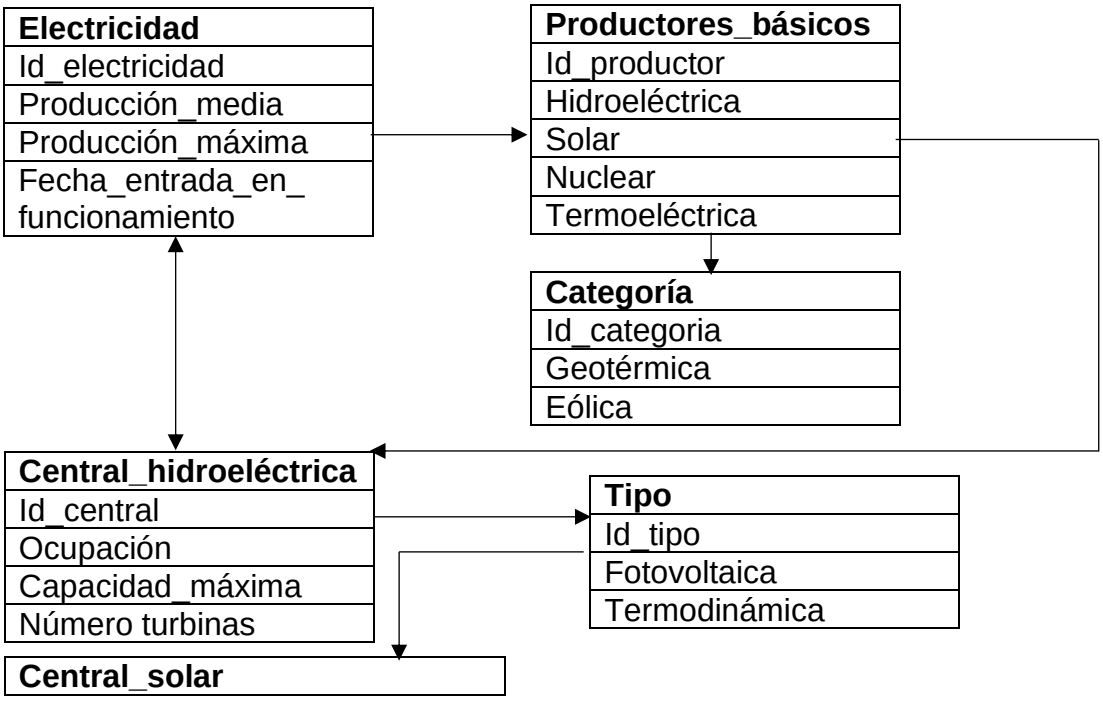
1.1 Modelo Conceptual - 15 puntos (GitBranch: feature/dba er)

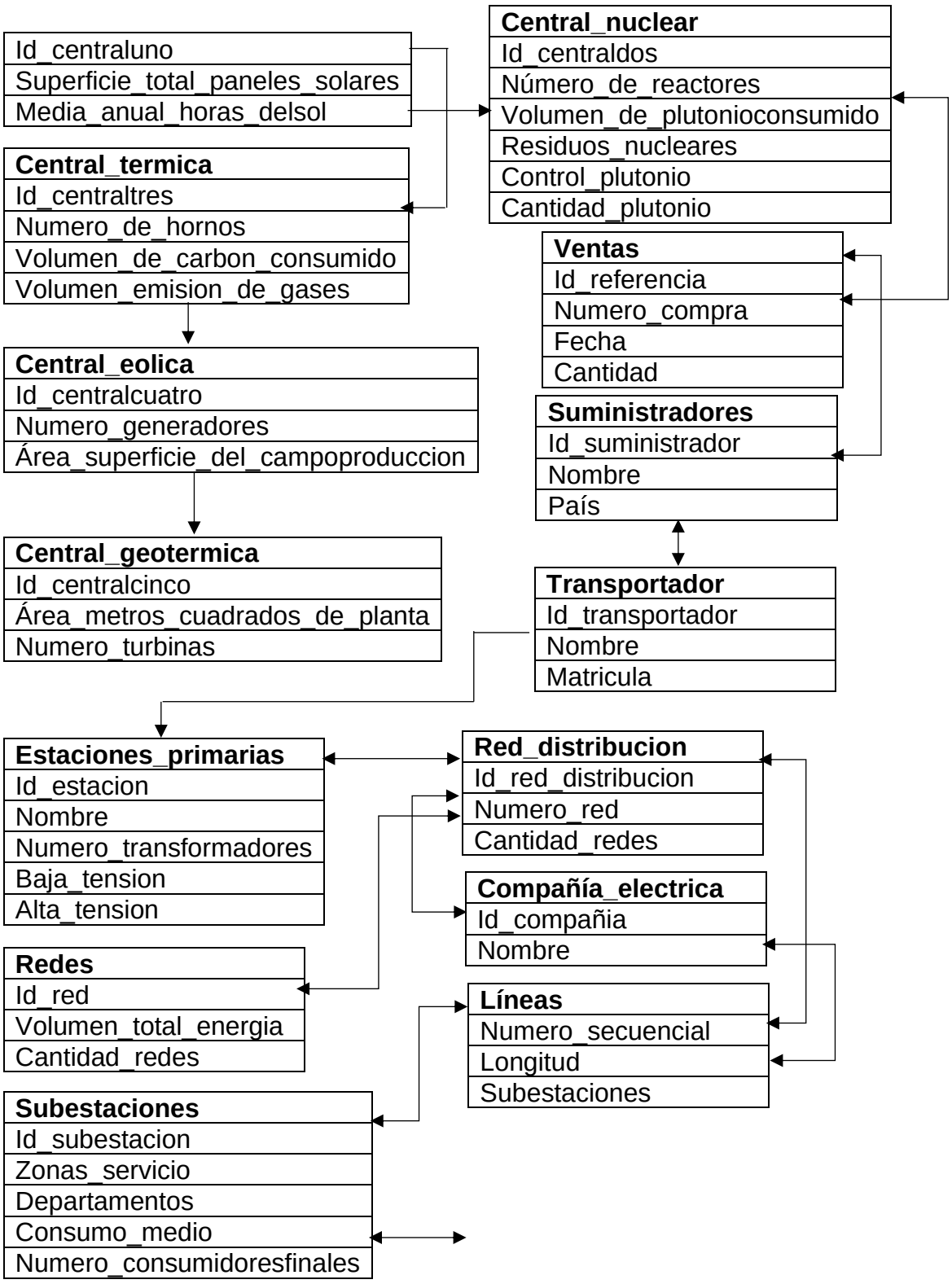
Realice el modelo ER2 del siguiente problema, donde se pretende registrar el modelo de negocio del sector energético. Se parte de las siguientes hipótesis:





1.2 Modelo Lógico - 10 puntos (GitBranch: feature/dba_r)




Id_centraluno
Superficie_total_paneles_solares
Media_anual_horas_delsol
Central_nuclear
Id_centraldos
Número_de_reactores
Volumen_de_plutonioconsumido
Residuos_nucleares
Control_plutonio
Cantidad_plutonio
Central_termica
Id_centraltres
Numero_de_hornos
Volumen_de_carbon_consumido
Volumen_emision_de_gases
Ventas
Id_referencia
Numero_compra
Fecha
Cantidad
Central_eolica
Id_centralcuatro
Numero_generadores
Área_superficie_del_campoproduccion
Suministradores
Id_suministrador
Nombre
País
Central_geotermica
Id_centralcinco
Área_metros_cuadrados_de_planta
Numero_turbinas
Transportador
Id_transportador
Nombre
Matricula
Estaciones_primarias
Id_estacion
Nombre
Numero_transformadores
Baja_tension
Alta_tension
Red_distribucion
Id_red_distribucion
Numero_red
Cantidad_redes
Compañía_electrica
Id_compañia
Nombre
Redes
Id_red
Volumen_total_energia
Cantidad_redes
Líneas
Numero_secuencial
Longitud
Subestaciones
Subestaciones
Id_subestacion
Zonas_servicio
Departamentos
Consumo_medio
Numero_consumidoresfinales

Cantidad_zonas

Categorías
Id_categoria
Particulares
Empresas
Instituciones
Cantidad_Particulares
Cantidad_Empresas
Cantidad_Instituciones

Bonus 5 puntos.
Implemente el modelo lógico en un motor relacional como SqlServer o Postgres.

1.3 SQL - 5 puntos
Escriba las siguientes consultas en SQL:

- Nombre, volumen de plutonio y residuos nucleares de productores de energía nuclear que suministren energía al menos tres países.
  Solución:
  Create database nuclear
  create table central_nuclear(
  id_central int NOT null AUTO_INCREMENT,
  PRIMARY KEY(id_central),
  Nombre varchar(20) not null,
  Volumen_plutonio varchar(100) not null,
  Residuos_nucleares varchar(20) not null,
  Paises varchar(20) not null
)
  Insert into central_nuclear values(1, ‘oro’, ‘100m3/’, ‘uranioradioactivo’, ‘Suiza’);
  Insert into central_nuclear values(2, ‘metales’, ‘50m3/’, ‘isotopos’, ‘Alemania’)
  Insert into central_nuclear values(3, ‘carbon’, ‘67m3/’, ‘uranioradioactivo’, ‘Francia’)

  Select * from central_nuclear

- Nombre, producción media, producción máxima, categoría del productor y cantidad de energía entregada a estaciones primarias para cada productor.
  Solución:
  Create database electricos
  create table electricidad(
  id_electricidad int NOT null AUTO_INCREMENT,
  PRIMARY KEY(id_electricidad),
  Nombre varchar(20) not null,
  Producción_media varchar(20) not null,

Producción_maxima varchar(20) not null,
Categoria_productor varchar(20) not null,
Cantidad_energia varchar(20) not null
)
Insert into electricidad values(1, ‘radioactividad’, ‘bajo’, ‘estable’, ‘A’, ‘12’);
Insert into electricidad values(2, ‘isotropico’, ‘alto’, ‘superior’, ‘C’, ‘6’);

Select * from electricidad

Bonus - 5 puntos (GitBranch: feature/dba sql bonus)

- Nombre de cada productor, estación primaria, red de distribución, subestación y zona con la cantidad de consumidores y consumo total de ellos.

Solución:
Create database productores
create table productores_generales(
id_productor int NOT null AUTO_INCREMENT,
PRIMARY KEY(id_productor),
Nombre_productor varchar(20) not null,
Estacion_primaria varchar(20) not null,
Red_distribucion varchar(20) not null,
Subestacion varchar(20) not null,
Cantidad_consumidores tinyint(100) not null,
Consumo_total bigint(100) not null
)
Insert into productores_generales values(1, ‘postobon’, ‘San Miguel’, ‘red norte’, ‘3’, ‘12’, ‘30’);
Insert into productores_generales values(2, ‘Bavaria’, ‘San Juan’, ‘red sur’, ‘6’, ‘14’, ‘80’);

Select * from productores_generales

- Consumo total y cantidad de consumidores de energía producida por termoeléctricas o hidroeléctricas por departamento.

Solución:
Create database energia
create table consumidores(
id_consumidor int NOT null AUTO_INCREMENT,
PRIMARY KEY(id_consumidor),
Consumo_total bigint(100) not null,
Cantidad_consumidores tinyint(20) not null,

departamento varchar(20) not null
)
Insert into consumidores values(1, ‘30’, ‘4’, ‘Armenia’);
Insert into consumidores values(2, ‘45’, ‘12’, ‘Cúcuta’);

Select * from consumidores

2. Arquitectura de Software
2.1. Modelado UML - 10 puntos (GitBranch: feature/back uml)

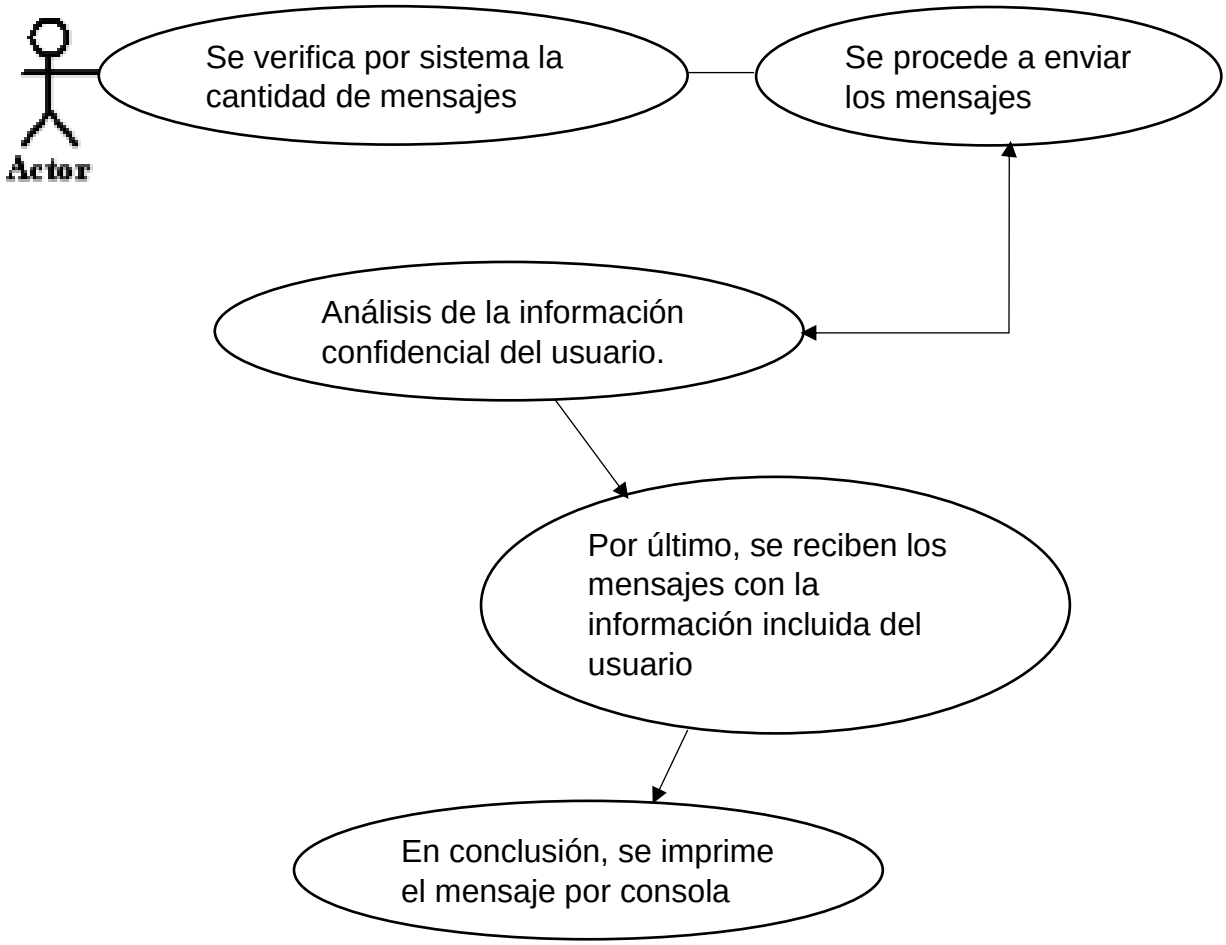
Diagrama de clases: a continuación, se evidencia un ejemplo de un diagrama de clases teniendo en cuenta el modelo UML.

Mensajes
Cantidad de mensajes. Registro de mensajes. Información confidencial. Datos del usuario.
Enviar mensajes () Recibir mensajes () Verificar si por sistema llegó o no el mensaje () Impresión de mensajes ()

Diagramas de objetos: se tiene el diagrama de objetos para la clase de mensajes, sin embargo, va vinculada al diagrama de clases

Transmisión_mensajes: Mensajes
Int cantidad = 0 String registro_mensajes = “enviar” String información = “informacion” String datos_usuario = “datos usuarios”

Diagramas de casos de uso: Los diagramas de caso de uso modelan la funcionalidad del sistema usando actores y casos de uso. Los casos de uso son servicios o funciones provistas por el sistema para sus usuarios.



Teniendo en cuenta los diagramas para UML se puede verificar la transmisión de mensajes por sistema, ciertamente los registros quedan incorporados en las bases de datos, se siguió su proceso de como se hace la ejecución de la emisión de mensajes, es decir son pasos necesarios para poder identificar la cantidad de registros enviados, quienes los enviaron y saber si llegaron o no dichos mensajes y también analizar si fue exitoso o fallido la operación.

Implementar el código en Java o Python.

```
public class mensajes{

private int cantidad_mensajes;
private String registro_mensajes;
private String informacion;
private String datos_usuarios;
```

```
public mensajes (int mens, String reg_mensajes, String info, String dat_usuarios){
cantidad_mensajes = mens;
registro_mensajes = reg_mensajes;
información = info;
datos_usuarios = dat_usuarios;
}
public void visualizarcantidad() {
System.out.println(“ en sistema hay ” + cantidad_mensajes + “mensajes”);
}
public void registrar(){
System.out.println(“ Se registraron ” + registro_mensajes + “mensajes por medio de las bases de datos”);
}
public void visualizarinformacion(){
System.out.println(“ Dicha información queda almacenada por el sistema base de datos ya que es una información sumamente confidencial”);
}
public void visualizardatos(){
System.out.println(“los datos del usuario fueron introducidos exitosamente”);
}
}

public class main{
public static void main(String[]args){
mensajes message = new mensajes (3, “24”, “Información confidencial de Miguel”, “enviar”)
message.visualizarcantidad();
message.registrar();
message.visualizarinformacion();
message.visualizardatos();
}
}
```

2.2. Patrones - 10 puntos (GitBranch: feature/back pattern)

Describa brevemente la utilidad de los siguientes patrones e implemente un ejemplo sencillo en lenguaje Python que refleje lo descrito.

- Observable.
- Decorador.
- Singleton.

Observable: el Patrón Observador (Pattern Observer), permiten contar con una amplia gama de soluciones a problemas habituales que se presentan en el diseño de softwares. Estos se pueden clasificar en tres grandes grupos, los patrones creacionales, estructurales y de comportamiento.

En código Python queda de la siguiente manera:

```
class Observable(object):
        def __init__(self, value):
            self.value = value
            self.observers = []

        def set(self, value):
            old = self.value
            self.value = value
            self.notifyObservers(old, self.value)

        def get(self):
            return self.value

        def addObserver(self, o):
            self.observers.append(o)

        def removeObserver(self, o):
            if o in self.observers:
                self.observers.remove(o)

        def notifyObservers(self, old, new):
            for o in self.observers:
                o.valueChanged(old, new)
```

Tomado del enlace: https://stackoverrun.com/es/q/11216858

Tal como lo dicen sus nombres, los patrones creacionales se refieren a los aspectos de creación o iniciación de los objetos; los estructurales se ocupan de los componentes que dan forma a los objetos y finalmente los de comportamiento se refieren a la manera en que interactúan los objetos. Dentro de este último grupo podemos encontrar el patrón Observer o Patrón Observador.

Decorador: El patrón decorador se compone principalmente de una Interfaz de la cual se implementa la clase concreta y los decoradores que añadirá mayor funcionalidad a la clase concreta.

En código Python queda de la siguiente manera:

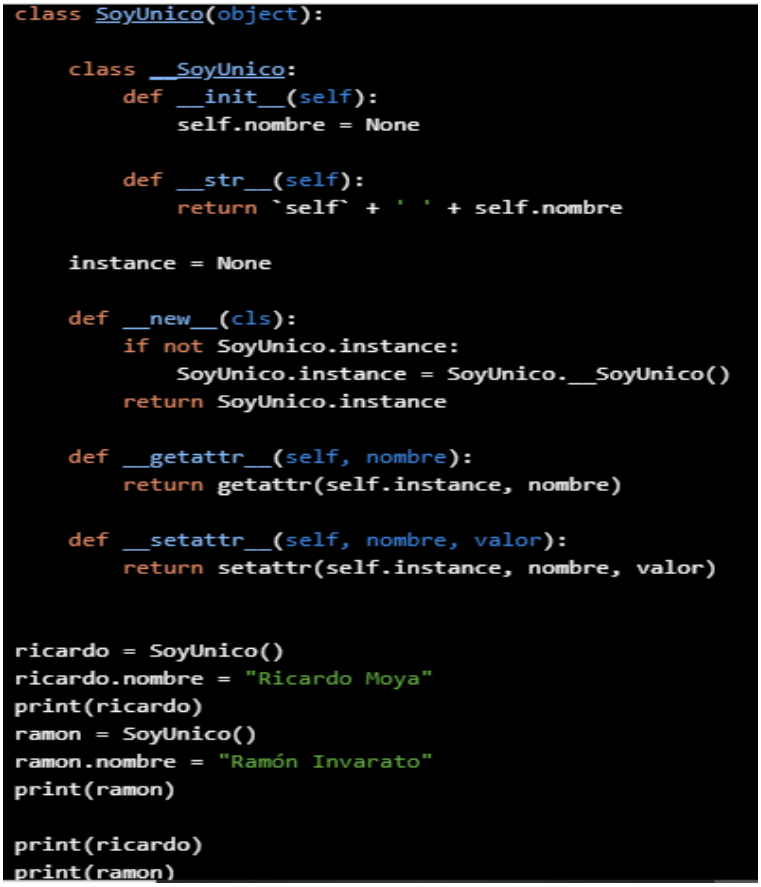
```python
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import random

class forat_negre_sonat(object):
    "Un decorador amb fam"
    def __init__(self, mostrar):
        self.mostrar = mostrar

    def __call__(self, f):
        def none(*args, **kw_args):
            if self.mostrar():
                return f(*args, **kw_args)
            else:
                return "Nop"
        return none

@forat_negre_sonat(mostrar = lambda :random.choice((True, False)))
def suma(a, b):
    "Suma dos elements que li passam com a paràmetre"
    return a+b

if __name__=="__main__":
    print suma(2,3)
    print suma(5,6)
    print suma(9,5)
```

Tomado del enlace: https://www.apsl.net/blog/2009/08/31/decoradores-en-python/

Singleton: El patrón de diseño Singleton (soltero) recibe su nombre debido a que sólo se puede tener una única instancia para toda la aplicación de una determinada clase, esto se logra restringiendo la libre creación de instancias de esta clase mediante el operador new e imponiendo un constructor privado y un método estático para poder obtener la instancia.

La intención de este patrón es garantizar que solamente pueda existir una única instancia de una determinada clase y que exista una referencia global en toda la aplicación.

En código Python queda de la siguiente manera:



```python
class SoyUnico(object):

    class __SoyUnico:
        def __init__(self):
            self.nombre = None

        def __str__(self):
            return `self` + ' ' + self.nombre

    instance = None

    def __new__(cls):
        if not SoyUnico.instance:
            SoyUnico.instance = SoyUnico.__SoyUnico()
        return SoyUnico.instance

    def __getattr__(self, nombre):
        return getattr(self.instance, nombre)

    def __setattr__(self, nombre, valor):
        return setattr(self.instance, nombre, valor)


ricardo = SoyUnico()
ricardo.nombre = "Ricardo Moya"
print(ricardo)
ramon = SoyUnico()
ramon.nombre = "Ramón Invarato"
print(ramon)

print(ricardo)
print(ramon)
```

Tomado del enlace: https://jarroba.com/patron-singleton-python-ejemplos/

3. Microservicios

3.1. GraphQL - 25 puntos (GitBranch: feature/api gql)

basesdatoselectricidad.py × | Python - Get Started

C: > Users > User > Desktop > visual_studio_code > basesdatoselectricidad.py

```python
import sqlite3
#conexion sqlite3
con = sqlite3.connect("ng.s3db")
cur = con.cursor()

#insertar
electricidad = ('bajo', 'estable', '2010/04/21')
cur.execute('insert into electricidad(Produccion_media, Produccion_maxima, fecha_entrada_en_funcionamiento) values (?,?,?,?)', electricidad)
con.commit()

productores_basicos = ('leve', 'alta', 'tipo A', 'quimica')
cur.execute('insert into productores_basicos(hidroelectrica, solar, nuclear, termoelectrica) values (?,?,?,?)', productores_basicos)
con.commit()

central_hidroelectrica = ('control de resistencia', '60%', '4')
cur.execute('insert into central_hidroelectrica(ocupacion, capacidad_maxima, numero_turbinas) values (?,?,?,?)', central_hidroelectrica)
con.commit()

central_solar = ('40 mts de longitud', '90 minutos')
cur.execute('insert into central_solar(superficie_total_paneles_solares, media_anual_horas_delsol) values (?,?,?,?)', central_solar)
con.commit()

central_nuclear = ('3', '40m3', 'materialradioactivo', '30%', '12')
cur.execute('insert into central_nuclear(numero_de_reactores, volumen_de_plutonioconsumido','residuos_nucleares, control_plutonio, cantidad-plutonio) values (?,?,?,?)', central_nuclear)
con.commit()

central_termica = ('5', '60m3', '30m3')
cur.execute('insert into central_termica(numero_de_hornos, volumen_de_carbon_consumido, volumen_emision_de_gases) values (?,?,?,?)', central_termica)
con.commit()
```