“Año del Bicentenario, de la consolidación de nuestra Independencia, y de la conmemoración de las heroicas batallas de Junín y Ayacucho”

CURSO:

Programación Orientada A Objetos

TEMA:

Clases persistentes con Python, C#, Scala

TEACHER:

Evelyn Marilyn Riveros Huaman

ALUMNOS:

MIGUEL ANGEL VELASQUEZ YSUIZA	U23231519
ROBERTO AGUSTÍN MEJÍA COLLAZOS	U23254461
MANUEL ÁNGEL PECHO SANTOS	U23201694
IVAN SULCA PALACIOS	U23220872
JOSUÉ CARMELO MURGA GUIMARAY	U23238660

Clases persistentes

1. Definición

Persistencia: Se denomina "persistencia" de los objetos a su capacidad para guardarse y recuperarse desde un medio de almacenamiento. Las clases persistentes son clases cuyas instancias pueden ser guardadas en una forma de almacenamiento persistente, como una base de datos o un archivo, y luego recuperadas y utilizadas más tarde.

Mapeo objeto-relacional (ORM): Una técnica que permite a los desarrolladores trabajar con bases de datos utilizando clases de programación orientada a objetos.

Serialización: El proceso de convertir un objeto en una forma que pueda ser almacenada y luego reconstruida.

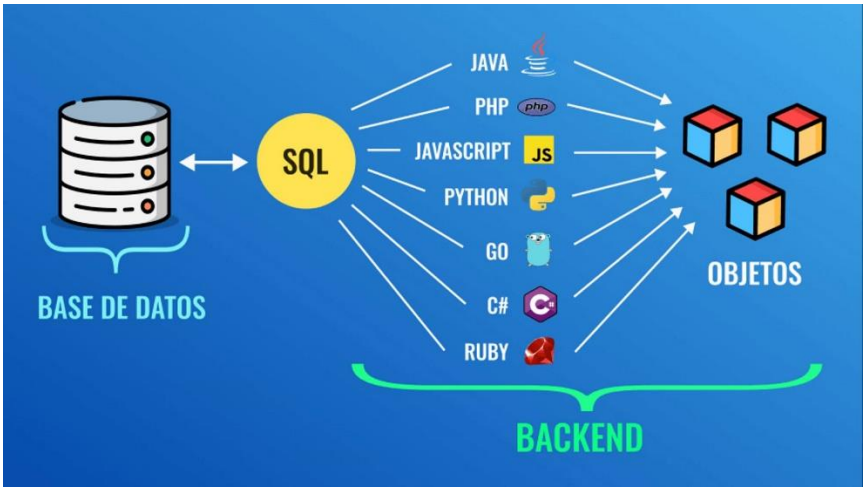
DML: Del inglés Data Manipulation Language, es un lenguaje proporcionado por los sistemas gestores de base de datos. Este lenguaje permite a los usuarios introducir datos para posteriormente realizar tareas de consultas o modificación.

2. Usos

Las clases persistente son clases en una aplicación que nos van a servir para representar entidades de la base de datos. Las clases persistentes tienen la capacidad de definir objetos que pueden almacenarse y recuperarse y un almacén persistente de datos.

3. Aplicación

Su aplicación se ha extendido en la mayoría de los lenguajes de programación, simplificando la persistencia de datos. Así también, facilita el mantenimiento de los registros debido a que evita el uso de sentencias DML para el mantenimiento de los registros. En lugar de ello se usa el lenguaje de programación, como Java, Python, C#, PHP, Scala, mediante funciones, clases, métodos, etc., los cuales realizan el mantenimiento, creando las sentencias DML a nivel interno para la comunicación con la base de datos.



Fuente: https://ed.team/blog/que-es-un-orm-y-como-funciona

a. Python

Para crear una clase persistente en Python se requiere de una librería ORM como el SQLAlchemy. Para este trabajo se ha instalado la versión 1.4.23 mediante el comando

pip install sqlalchemy==1.4.23

Código

En el archivo *base.py* se declara la base de datos

```
from sqlalchemy.ext.declarative import declarative_base

# Crear la base de datos base
Base = declarative_base()
```

En el archivo *alumno.py* se declara la clase persistente que hereda de Base

```
from sqlalchemy import Column, Integer, String
from base import Base

# Definir la clase persistente
class Alumno(Base):
    __tablename__ = 'alumno'

    id = Column(Integer, primary_key=True, autoincrement=True)
    nombre = Column(String, nullable=False)
    codigo = Column(String, nullable=False)

    def __repr__(self):
        return f"<Alumno(codigo='{self.codigo}',
nombre='{self.nombre}')>"
```

En el archive *engine.py* se configura la conexión a la base de datos. En este caso será un archivo con nombre “database.db”

```
from sqlalchemy import create_engine

engine = create_engine('sqlite:///database.db', echo=True)
```

Luego de configurar la conexión a la base de datos y establecer la clase persistente, se ejecutará el siguiente archivo *migrate.py* para crear la estructura de la tabla "Alumno" en la base de datos

```python
from base import Base
from engine import engine

# Cargar las clases
from alumno import Alumno

# Crear las tablas en la base de datos
Base.metadata.create_all(engine)
```

python.exe .\migrate.py

El archive *session.py* crea una sesión de conexión a la base de datos *database.db*

```python
from sqlalchemy.orm import sessionmaker

from engine import engine

# Crear una sesión
Session = sessionmaker(bind=engine)
session = Session()
```

Para llenar datos a la tabla se creó el archivo *seed.py* el cual inserta datos de alumnos a la tabla Alumno

```python
from session import session
from alumno import Alumno

# Crear una nueva persona
alumno = Alumno(nombre="Josué Murga", codigo="U23238660")

# Agregar la persona a la sesión
session.add(alumno)
session.add(Alumno(nombre="Miguel Velazquez", codigo="U23231519"))
session.add(Alumno(nombre="Roberto Mejía", codigo="U23254461"))
session.add(Alumno(nombre="Manuel Pecho", codigo="U23201694"))
session.add(Alumno(nombre="Iván Sulca", codigo="U23220872"))

# Confirmar la transacción
session.commit()
```

python.exe .\seed.py

Para insertar datos desde consola, se puede ejecutar el archivo *insert.py*

```python
from alumno import Alumno
from session import session

codigo = input("Ingrese código:\n")
nombre = input("Ingrese nombre:\n")

# Crear una nueva persona
alumno = Alumno(nombre=nombre, codigo=codigo)

# Agregar la persona a la sesión
session.add(alumno)

# Confirmar la transacción
session.commit()
```

python.exe .\insert.py

```
Ingrese código:
U12345678
Ingrese nombre:
Juan Perez
2024-06-25 23:32:06,610 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2024-06-25 23:32:06,611 INFO sqlalchemy.engine.Engine INSERT INTO alumno (nombre, codigo) VALUES (?, ?)
2024-06-25 23:32:06,611 INFO sqlalchemy.engine.Engine [generated in 0.00021s] ('Juan Perez', 'U12345678')
2024-06-25 23:32:06,613 INFO sqlalchemy.engine.Engine COMMIT
```

El archivo *listar.py* lista los datos de los alumnos que han se han guardado en la base de datos usando la clase persistente Alumno

```python
from alumno import Alumno
from session import session

# Consultar la base de datos para recuperar todas las personas
alumnos = session.query(Alumno).all()

# Imprimir los resultados
for person in alumnos:
    print(person)
```

python.exe .\list.py

```
2024-06-25 23:37:06,733 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2024-06-25 23:37:06,734 INFO sqlalchemy.engine.Engine SELECT alumno.id AS alumno_id, alumno.nombre AS alumno_nombre, alumno.codigo AS alumno_codigo
FROM alumno
2024-06-25 23:37:06,734 INFO sqlalchemy.engine.Engine [generated in 0.00016s] ()
<Alumno(codigo='U23238660', nombre='Josué Murga')>
<Alumno(codigo='U23231519', nombre='Miguel Velazquez')>
<Alumno(codigo='U23254461', nombre='Roberto Mejía')>
<Alumno(codigo='U23201694', nombre='Manuel Pecho')>
<Alumno(codigo='U23220872', nombre='Iván Sulca')>
<Alumno(codigo='U12345678', nombre='Juan Perez')>
```

b. C#

Para desplegar una aplicación con clases persistentes en el lenguaje C# se usará el framework .NET y el ORM EntityFramework. Estas librerías se instalan mediante los siguientes comandos

dotnet add package Microsoft.EntityFrameworkCore --version 7.0.0

dotnet add package Microsoft.EntityFrameworkCore.Sqlite --version 7.0.0

dotnet add package Microsoft.EntityFrameworkCore.Tools --version 7.0.0

Código

Crear la clase Alumno en el archivo *Alumno.cs*.

```
public class Alumno
{
    public int Id { get; set; }
    public required string Codigo { get; set; }
    public string? Nombre { get; set; }
}
```

Se procede a crear la conexión a la base de datos en el archivo *AppDbContext.cs*.

```
using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext
{
    public DbSet<Alumno> Alumnos { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=app.db");
    }
}
```

La codificación para instanciar una clase persistente es muy similar al de una clase normal.

```
            // Crear un nuevo alumno
            var alumno = new Alumno { Nombre = "Josué Murga", Codigo =
"U23238660" };
```

El siguiente archivo inserta un registro con la clase persistente, luego muestra los datos registrados

```
using System;
using System.Linq;

class Program
{
    static void Main(string[] args)
    {
        using (var context = new AppDbContext())
        {
            // Crear un nuevo alumno
            var alumno = new Alumno { Nombre = "Josué Murga", Codigo =
"U23238660" };

            // Agregar la persona al contexto y guardar cambios
            context.Alumnos.Add(alumno);
            context.SaveChanges();

            // Consultar la base de datos para recuperar todas las
personas
            var alumnos = context.Alumnos.ToList();

            // Imprimir los resultados
            foreach (var a in alumnos)
            {
                Console.WriteLine($"ID: {a.Id}, Código: {a.Codigo},
Nombre: {a.Nombre}");
            }
        }
    }
}
```

El siguiente comando es para crear el código para la generación de la base de datos

dotnet ef migrations add InitialCreate

Para proceder a crear la estructura de la base de datos se ejecuta el siguiente comado

dotnet ef database update

Para proceder a insertar el registro, luego listarlo se ejecuta:

dotnet run

```
PS C:\Users\Josue\Projects\C#\clase_persistente\TareaAcademicaClasePersistente> dotnet run
ID: 1, Código: U23238660, Nombre: Josué Murga
```

c. Scala

En el caso del lenguaje escala, se usa la librería Slick, el cual es un ORM que permite la codificación de clases persistentes.

Código

Se configura el archivo build.sbt con las librerías necesarias

```
name := "SlickExample"

version := "1.0"

scalaVersion := "2.13.5"

libraryDependencies ++= Seq(
  "com.typesafe.slick" %% "slick" % "3.3.3",
  "com.typesafe.slick" %% "slick-hikaricp" % "3.3.3", // Para la gestión
de la conexión
  "org.xerial" % "sqlite-jdbc" % "3.36.0.3"
)

mainClass in Compile := Some("example.Main")
```

Se crean los archivos Main.scala y Models.scala en la carpeta /src/main/scala/example

El archivo Models.scala contendrá la clase persistente

```
package example

import slick.jdbc.SQLiteProfile.api._

// Define la entidad persistente
case class Alumno(id: Option[Int], codigo: String, nombre: String)

// Define la tabla correspondiente en la base de datos
class Alumnos(tag: Tag) extends Table[Alumno](tag, "alumnos") {
  def id = column[Int]("id", O.PrimaryKey, O.AutoInc)
  def codigo = column[String]("codigo")
  def nombre = column[String]("nombre")

  def * = (id.?, codigo, nombre) <> (Alumno.tupled, Alumno.unapply)
}

object Alumnos {
  val table = TableQuery[Alumnos]
}
```

El siguiente código primeramente crea la estructura en la base de datos si no ha sido creada. Luego registra un alumno usando la clase persistente. Finalmente, procede a listar los registros que persistan en la tabla.

```
package example

import slick.jdbc.SQLiteProfile.api._
import scala.concurrent.Await
import scala.concurrent.duration._
import scala.concurrent.ExecutionContext.Implicits.global

object Main extends App {
  // Crea la conexión a la base de datos
  val db = Database.forConfig("sqliteDB")

  // Crear la tabla si no existe
  try {
    val setupAction = Alumnos.table.schema.create
    val setupResult = db.run(setupAction)
    Await.result(setupResult, 10.seconds)
  } catch {
    case _: Throwable => println("La tabla ya existe.")
  }

  // Insertar un alumno
  val insertAction = Alumnos.table += Alumno(None, "U23238660", "Josue
Murga")
  val insertResult = db.run(insertAction)
  Await.result(insertResult, 10.seconds)

  // Consultar todas los alumnos
  val queryAction = Alumnos.table.result
  val queryResult = db.run(queryAction)
  val alumnos = Await.result(queryResult, 10.seconds)

  // Imprimir los resultados
  alumnos.foreach(println)

  // Cerrar la conexión a la base de datos
  db.close()
}
```

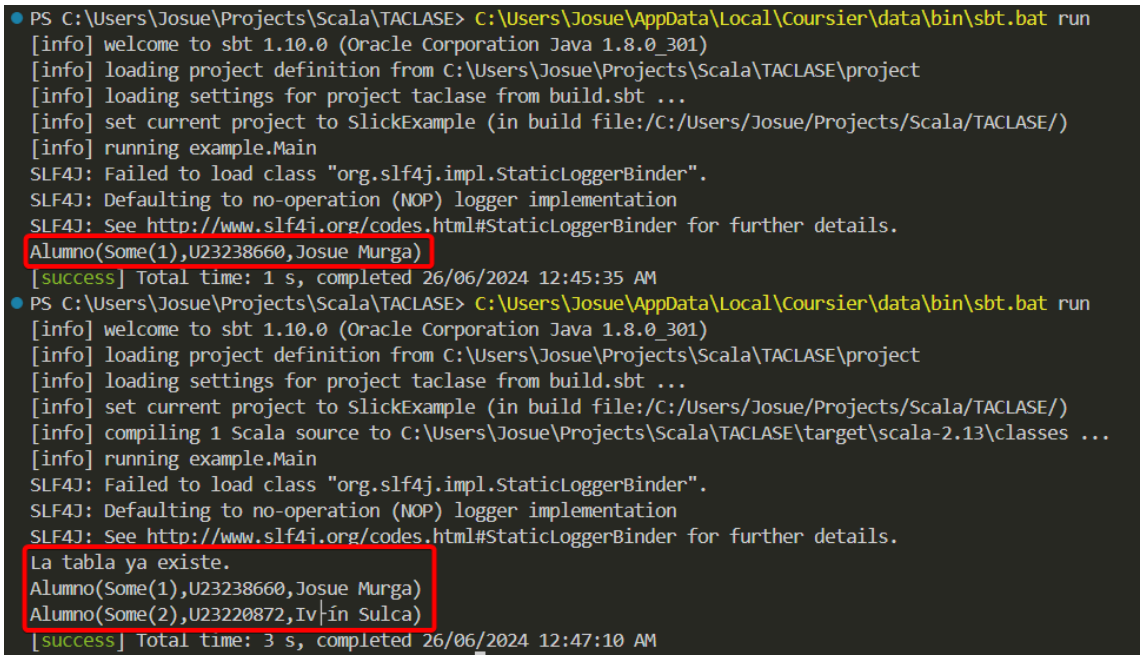
Se ejecuta con los siguientes comandos

sbt update

sbt compile

sbt run

Ejecución del código donde se puede apreciar el registro del primer alumno en el primer recuadro rojo. En el segundo recuadro el sistema indica que la estructura de tablas ya ha sido creada, por lo que solo procede a insertar y listar los registros.



Referencias

S’ Arreplec (s. f.) *Clases persistentes.* Recuperado el 25 de junio de 2024 de https://sarreplec.caib.es/pluginfile.php/9710/mod_resource/content/7/AD04_Contenidos_Web/7_clases_persistentes.html

Segovia, José (07 de junio de 2017) *Diferencias entre DDL, DML y DCL.* TodoPostgreSQL. Recuperado el 26 de junio de 2024 de https://www.todopostgresql.com/diferencias-entre-ddl-dml-y-dcl/