

CURSO : Desarrollo de Aplicaciones Web I (0265)  
PROFESOR : César Enrique Santos Torres  
CICLO : Quinto  
SECCIÓN : 28257  
GRUPO : 2024331933  
FECHA : 23/11/2024 10:00am  
DURACIÓN : 2 horas

NOTA

ALUMNO (A) : Jesus Medina Lazaro

### CASO DE LABORATORIO 1 (CL1)

#### Consideraciones generales:

- El laboratorio consta de 4 partes, cada parte tiene una secuencia de pasos las cuales deberá ir acompañada (De forma obligatoria) de capturas de pantalla de lo implementado.
- Sólo debe subir este documento, con sus evidencias y respuestas en él. El código fuente de ambos proyectos debe ser subido a Github (Adjuntar links del repositorio). No se aceptará código zipeado.
- El nombre del presente archivo deberá tener la siguiente estructura: "DAWI-APELLIDOS-NOMBRES.pdf".

#### LOGRO DE LA EVALUACIÓN:

Al término de la evaluación, el alumno implementa las operaciones de mantenimiento sobre una entidad utilizando Java Persistence API.

#### CONSOLIDADO

Pregunta	Puntaje		Llenar solo en caso de Recalificación justificada	
	Máximo	Obtenido	Sustento	Puntaje
1	5			
2	5			
3	5			
4	5			
Total	20			
Nota Recalificada				

## Parte 01 Configuración básica (25%)

- Descargar JDK versión 23 de <https://adoptium.net/es/temurin/releases/>
- Configurar variable de entorno JAVA\_HOME y Path
- Validar configuración Java con los siguientes comandos (Use cmd):
  - java -version
  - echo %JAVA\_HOME%
  - echo %Path%

```
C:\Users\weikg>java --version
openjdk 23.0.1 2024-10-15
OpenJDK Runtime Environment Temurin-23.0.1+11 (build 23.0.1+11)
OpenJDK 64-Bit Server VM Temurin-23.0.1+11 (build 23.0.1+11, mixed mode, sharing)
```

- Conectar al servidor MySQL usando la terminal (Use cmd):
  - Use el comando: mysql -u root -p

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

- Restaurar bd “world” de <https://downloads.mysql.com/docs/world-db.zip> (Use cmd):
  - Use el comando: source <ruta-archivo-world.sql>;Ejecute este comando:
  - source D:\weikg\Descargas\world-db\world-db\world.sqlMe salió este resultado:

```
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
mysql> |
```

- Usar bd “world” y hacer un select de los primeros 20 registros de la tabla “city” (Use cmd).

```
mysql> use `world`;
Database changed
mysql> select * from `city` LIMIT 20;
```

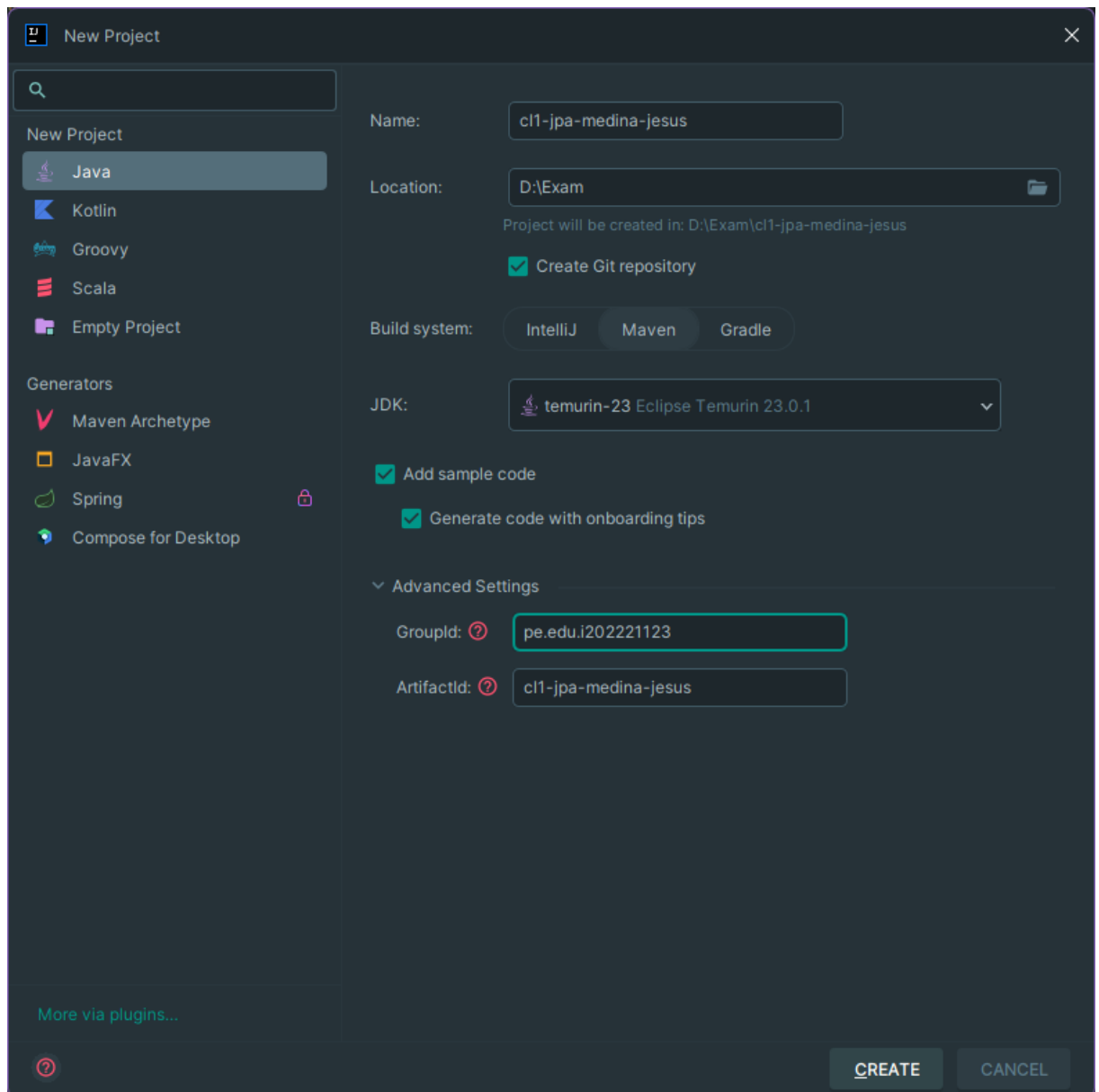
ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
7	Haag	NLD	Zuid-Holland	440900
8	Utrecht	NLD	Utrecht	234323
9	Eindhoven	NLD	Noord-Brabant	201843
10	Tilburg	NLD	Noord-Brabant	193238
11	Groningen	NLD	Groningen	172701
12	Breda	NLD	Noord-Brabant	160398
13	Apeldoorn	NLD	Gelderland	153491
14	Nijmegen	NLD	Gelderland	152463
15	Enschede	NLD	Overijssel	149544
16	Haarlem	NLD	Noord-Holland	148772
17	Almere	NLD	Flevoland	142465
18	Arnhem	NLD	Gelderland	138020
19	Zaanstad	NLD	Noord-Holland	135621
20	's-Hertogenbosch	NLD	Noord-Brabant	129170

```
20 rows in set (0.00 sec)

mysql> |
```

## Parte 02 Proyecto JPA-Hibernate (25%)

- Crear un proyecto JPA-Hibernate desde IntelliJ Idea con las siguientes características:
  - **Name:** cl1-jpa-<apellidoPaterno-primerNombre> (Use minúsculas y guión "-")
  - **Location:** Seleccione un directorio con permisos de lectura y escritura
  - **Create Git repository:** Check
  - **Build system:** Maven
  - **JDK:** Eclipse Temurin-23
  - **Advanced Settings:**
    - **GroupId:** pe.edu.<codigoEstudiante>
    - **Artifact:** cl1-jpa-<apellidoPaterno-primerNombre>



Configurar dependencias en el pom.xml

- Hibernate (6.6.2.Final)
- Driver de MySQL (9.1.0)

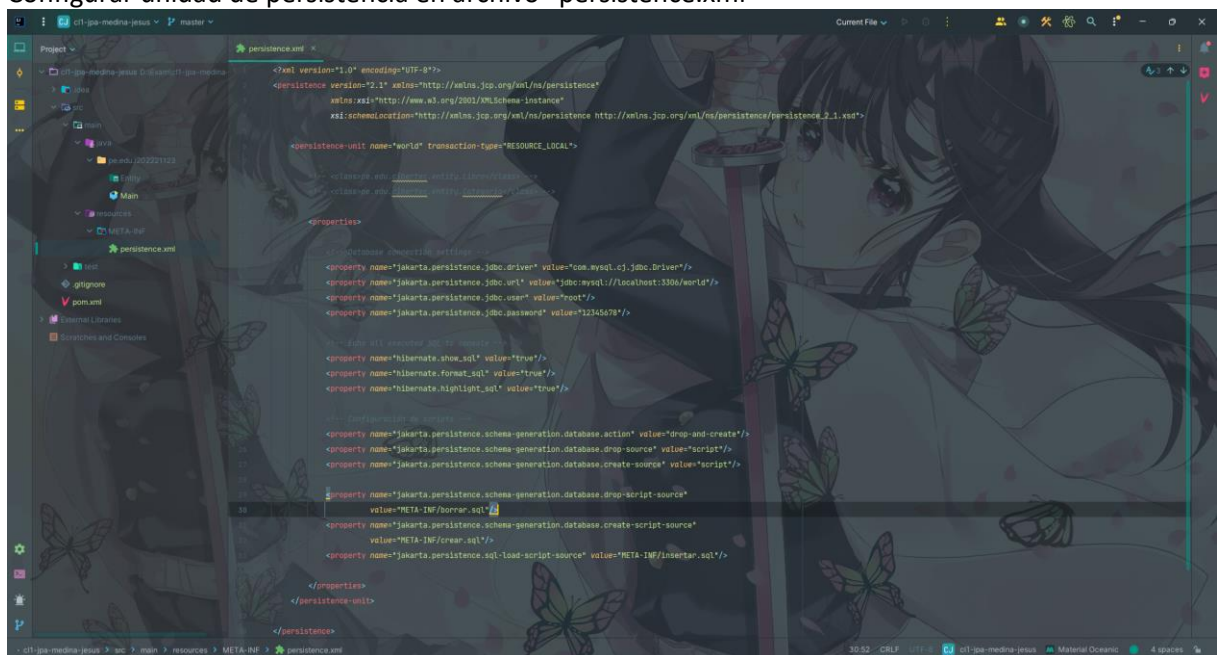
```

<dependencies>
  <!-- Hibernate -->
  <dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.6.2.Final</version>
  </dependency>

  <!-- MySQL Driver -->
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>9.1.0</version>
  </dependency>
</dependencies>

```

- 
- Configurar unidad de persistencia en archivo “persistence.xml”



The screenshot shows an IDE with a project named 'citi-gpa-medina-jesus'. The 'persistence.xml' file is open, showing the following configuration:

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2003/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">

  <persistence-unit name="world" transaction-type="RESOURCE_LOCAL">

    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <class>com.mysql.jdbc.Driver</class>

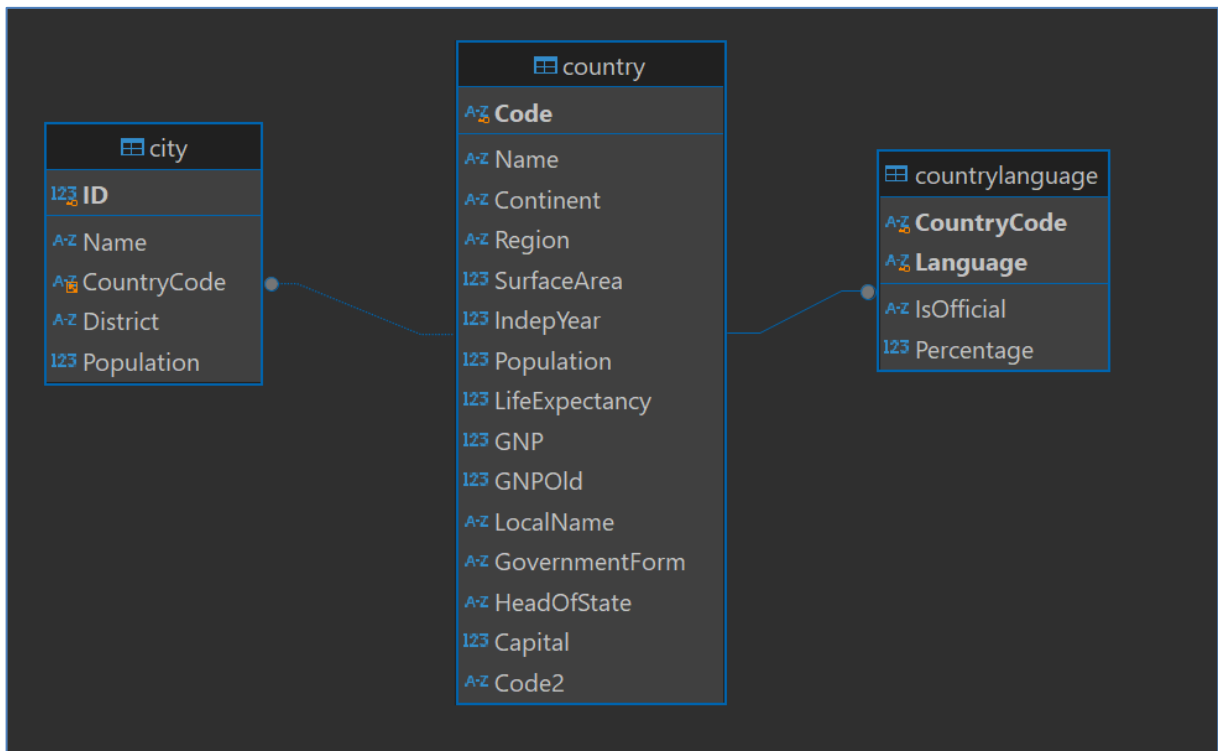
    <properties>
      <!-- Database (Hibernate) properties -->
      <property name="jakarta.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
      <property name="jakarta.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/world"/>
      <property name="jakarta.persistence.jdbc.user" value="root"/>
      <property name="jakarta.persistence.jdbc.password" value="12345678"/>

      <!-- JPA all resources JPA to resource -->
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.hbm2ddl.auto" value="true"/>

      <!-- JPA persistence unit properties -->
      <property name="jakarta.persistence.schema-generation.database.action" value="drop-and-create"/>
      <property name="jakarta.persistence.schema-generation.database.drop-source" value="script"/>
      <property name="jakarta.persistence.schema-generation.database.create-source" value="script"/>
      <property name="jakarta.persistence.schema-generation.database.create-script-source"
        value="META-INF/orm.xml"/>
      <property name="jakarta.persistence.schema-generation.database.create-script-source"
        value="META-INF/orm.xml"/>
      <property name="jakarta.persistence.sql-load-script-source" value="META-INF/load-orm.xml"/>
    </properties>
  </persistence-unit>
</persistence>

```

- Crear las entidades correspondientes a las siguientes tablas de la bd “world”:



- Mapear las 3 entidades de forma tradicional (Sin Lombok).
  - Definir la estrategia de generación correcta para los PKs.
  - Considerar el mapeo de las relaciones de forma bidireccional.
- Country**

```

1 package pe.edu.i202221123.Entity;
2
3 import jakarta.persistence.*;
4
5 import java.util.Set;
6
7 @Entity @usages new *
8 @Table(name = "country")
9 public class Country {
10     @Id @usages
11     @Column(name = "Code")
12     private String code; // Clave primaria
13
14     @Column(name = "Name") @usages
15     private String name;
16
17     @Column(name = "Continent") @usages
18     private String continent;
19
20     @Column(name = "Region") @usages
21     private String region;
22
23     @Column(name = "SurfaceArea") @usages
24     private Double surfaceArea;
25
26     @Column(name = "IndepYear") @usages
27     private Integer independYear;
28
29     @Column(name = "Population") @usages
30     private Integer population;
31
32     @Column(name = "LifeExpectancy") @usages
33     private Double lifeExpectancy;
34
35     @Column(name = "GNP") @usages
36     private Double gnp;
37
38     @Column(name = "GNPOld") @usages
39     private Double gnpoOld;
40
41     @Column(name = "LocalName") @usages
42     private String localName;
43
44     @Column(name = "GovernmentForm") @usages
45     private String governmentForm;
46
47     @Column(name = "HeadOfState") @usages
48     private String headOfState;
49
50     @Column(name = "Capital") @usages
51     private Integer capital;
52
53     @Column(name = "Code2") @usages
54     private String code2;
55
56     @OneToMany(mappedBy = "country") @usages
57     private Set<City> cities; // Relación bidireccional
58
59     @OneToMany(mappedBy = "country") @usages
60     private Set<CountryLanguage> countryLanguages;

```

```

9 public class Country {
10
11     public String getCode() { no usages new *
12         return code;
13     }
14
15     public void setCode(String code) { no usages new *
16         this.code = code;
17     }
18
19     public String getName() { no usages new *
20         return name;
21     }
22
23     public void setName(String name) { no usages new *
24         this.name = name;
25     }
26
27     public String getContinent() { no usages new *
28         return continent;
29     }
30
31     public void setContinent(String continent) { no usages new *
32         this.continent = continent;
33     }
34
35     public String getRegion() { no usages new *
36         return region;
37     }
38
39     public void setRegion(String region) { no usages new *
40         this.region = region;
41     }
42
43     public Double getSurfaceArea() { no usages new *
44         return surfaceArea;
45     }
46
47     public void setSurfaceArea(Double surfaceArea) { no usages new *
48         this.surfaceArea = surfaceArea;
49     }
50
51     public Integer getIndependYear() { no usages new *
52         return independYear;
53     }
54
55     public void setIndependYear(Integer independYear) { no usages new *
56         this.independYear = independYear;
57     }
58
59     public Integer getPopulation() { no usages new *
60         return population;
61     }
62
63     public void setPopulation(Integer population) { no usages new *
64         this.population = population;
65     }
66
67     public Double getLifeExpectancy() { no usages new *
68         return lifeExpectancy;
69     }
70
71     public void setLifeExpectancy(Double lifeExpectancy) { no usages new *
72         this.lifeExpectancy = lifeExpectancy;
73     }
74
75     public Double getGnp() { no usages new *
76         return gnp;
77     }
78
79     public void setGnp(Double gnp) { no usages new *
80         this.gnp = gnp;
81     }
82
83     public Double getGnpoOld() { no usages new *
84         return gnpoOld;
85     }
86
87     public void setGnpoOld(Double gnpoOld) { no usages new *
88         this.gnpoOld = gnpoOld;
89     }
90
91     public String getLocalName() { no usages new *
92         return localName;
93     }
94
95     public void setLocalName(String localName) { no usages new *
96         this.localName = localName;
97     }
98
99     public String getGovernmentForm() { no usages new *
100         return governmentForm;
101     }
102
103     public void setGovernmentForm(String governmentForm) { no usages new *
104         this.governmentForm = governmentForm;
105     }

```

City



```
persistence.xml Country.java City.java CountryLanguage.java
@Entity 4 usages new *
@Table(name = "city")
public class City {
    @Id 4 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-incremento por defecto
    @Column(name = "ID")
    private Integer id; // Clave primaria

    @Column(name = "Name") 4 usages
    private String name;

    @ManyToOne 4 usages
    @JoinColumn(name = "CountryCode", referencedColumnName = "Code")
    private Country country; // Relación bidireccional con la entidad Country

    @Column(name = "District") 4 usages
    private String district;

    @Column(name = "Population") 4 usages
    private Integer population;

    public City() {} no usages new *

    public City(Integer id, String name, Country country, String district, Integer
        this.id = id;
        this.name = name;
        this.country = country;
        this.district = district;
        this.population = population;
    }

    public Integer getId() { no usages new *
        return id;
    }

    public void setId(Integer id) { no usages new *
        this.id = id;
    }

    public String getName() { no usages new *
        return name;
    }

    public void setName(String name) { no usages new *
        this.name = name;
    }

    public Country getCountry() { no usages new *
        return country;
    }

    public void setCountry(Country country) { no usages new *
        this.country = country;
    }

    public String getDistrict() { no usages new *
        return district;
    }

    public void setDistrict(String district) { no usages new *
        this.district = district;
    }

    public Integer getPopulation() { no usages new *
        return population;
    }

    public void setPopulation(Integer population) { no usages new *
        this.population = population;
    }
}
```

CountryLanguage



```
persistence.xml Country.java City.java CountryLanguage.java x ↵
@Entity 4 usages new *
@Table(name = "countryLanguage")
public class CountryLanguage {
    @EmbeddedId 4 usages
    private CountryLanguageId id; // Clave primaria compuesta

    @ManyToOne 4 usages
    @JoinColumn(name = "CountryCode", referencedColumnName = "Code", insertable = false)
    private Country country; // Relación bidireccional con la entidad Country

    @Column(name = "Language") 4 usages
    private String language;

    @Column(name = "IsOfficial") 4 usages
    private String isOfficial;

    @Column(name = "Percentage") 4 usages
    private Double percentage;

    public CountryLanguage() {} no usages new *

    public CountryLanguage(CountryLanguageId id, Country country, String language, String isOfficial, Double percentage) {
        this.id = id;
        this.country = country;
        this.language = language;
        this.isOfficial = isOfficial;
        this.percentage = percentage;
    }

    public CountryLanguageId getId() { no usages new *
        return id;
    }

    public void setId(CountryLanguageId id) { no usages new *
        this.id = id;
    }

    public String getLanguage() { no usages new *
        return language;
    }

    public void setLanguage(String language) { no usages new *
        this.language = language;
    }

    public Country getCountry() { no usages new *
        return country;
    }

    public void setCountry(Country country) { no usages new *
        this.country = country;
    }

    public String getIsOfficial() { no usages new *
        return isOfficial;
    }

    public void setIsOfficial(String isOfficial) { no usages new *
        this.isOfficial = isOfficial;
    }

    public Double getPercentage() { no usages new *
        return percentage;
    }

    public void setPercentage(Double percentage) { no usages new *
        this.percentage = percentage;
    }
}
```

- Crear una clase “JPAPersist” y en ella registre un país imaginario, que tenga 3 ciudades y 2 lenguajes nativos. Sólo debe realizar un llamado al método “persist”.

	Capital	GNP	GNPOld	IndepYear	LifeExpectancy	Population	SurfaceArea	Code	Code2	Continent	GovernmentForm	HeadOfState	LocalName	Name	Region
▶	1	50000	NULL	2020	75	1000000	100000	IMN	IM	Imaginary Continent	Democracy	John Doe	Imaginary	Imaginary Nation	Unknown
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

	Percentage	IsOfficial	countryCode	language
▶	100	T	IMN	Imaginary Language 1
	50	F	IMN	Imaginary Language 2
*	NULL	NULL	NULL	NULL

	ID	Population	CountryCode	District	Name
▶	1	300000	IMN	District A	City One
	2	200000	IMN	District B	City Two
	3	100000	IMN	District C	City Three
*	NULL	NULL	NULL	NULL	NULL

```

13 public static void main(String[] args) { new *
14
15     try {
16
17         et.begin();
18
19
20
21         // Crear el país imaginario
22         Country country = new Country();
23         country.setCode("IMN");
24         country.setName("Imaginary Nation");
25         country.setContinent("Imaginary Continent");
26         country.setRegion("Unknown");
27         country.setSurfaceArea(100000.00);
28         country.setIndepYear(2020);
29         country.setPopulation(1000000);
30         country.setLifeExpectancy(75.0);
31         country.setGnp(50000.00);
32         country.setLocalName("Imaginary");
33         country.setGovernmentForm("Democracy");
34         country.setHeadOfState("John Doe");
35         country.setCapital(1);
36         country.setCode2("IM");
37
38
39         // Crear las ciudades
40         City city1 = new City();
41         city1.setName("City One");
42         city1.setDistrict("District A");
43         city1.setPopulation(300000);
44         city1.setCountry(country);
45
46
47         City city2 = new City();
48         city2.setName("City Two");
49         city2.setDistrict("District B");
50         city2.setPopulation(200000);
51         city2.setCountry(country);
52
53
54         City city3 = new City();
55         city3.setName("City Three");
56         city3.setDistrict("District C");
57         city3.setPopulation(100000);
58         city3.setCountry(country);
59
60
61         // Crear los lenguajes nativos
62         CountryLanguage language1 = new CountryLanguage();
63         CountryLanguageId languageId1 = new CountryLanguageId();
64         languageId1.setCountryCode("IMN");
65         languageId1.setLanguage("Imaginary Language 1");
66         language1.setCountryCode(languageId1);
67         language1.setIsOfficial("T");
68         language1.setPercentage(100.0);
69         language1.setCountry(country);
70
71
72         CountryLanguage language2 = new CountryLanguage();
73         CountryLanguageId languageId2 = new CountryLanguageId();
74         languageId2.setCountryCode("IMN");
75         languageId2.setLanguage("Imaginary Language 2");
76         language2.setCountryCode(languageId2);
77         language2.setIsOfficial("F");
78         language2.setPercentage(50.0);
79         language2.setCountry(country);
80
81
82         // Persistir el país, ciudades y lenguajes
83         en.persist(country);
84         en.persist(city1);
85         en.persist(city2);
86         en.persist(city3);
87         en.persist(language1);
88         en.persist(language2);
89
90         // Commit de la transacción
91         et.commit();
92     } catch (Exception e) {
93         et.rollback();
94     }
95 }

```

- Crear una clase “JPARemove” y en ella elimine el país imaginario (Previamente creado). La eliminación debe eliminar el rastro de sus 3 ciudades y 2 lenguajes nativos. Sólo debe realizar un llamado al método “remove”. Considere, no afectar la funcionalidad de la clase “JPAPersist”.

```

1 package pe.edu.i202221123.Crud;
2
3 import jakarta.persistence.*;
4 import org.hibernate.internal.build.AllowSysOut;
5 import pe.edu.i202221123.Entity.Country;
6
7 public class JPARemove {
8     public static void main(String[] args) {
9         EntityManagerFactory emf = Persistence.createEntityManagerFactory("world");
10        EntityManager em = emf.createEntityManager();
11
12        Country country = em.find(Country.class, "IMN");
13
14        em.getTransaction().begin();
15        em.remove(country);
16        em.getTransaction().commit();
17    }
18 }

```

	Capital	GNP	GNPOld	IndepYear	LifeExpectancy	Population	SurfaceArea	Code	Code2
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

	Percentage	IsOfficial	countryCode	language
*	NULL	NULL	NULL	NULL

	ID	Population	CountryCode	District	Name
*	NULL	NULL	NULL	NULL	NULL

- Crear una clase “JPAFind” y en ella realice una sola consulta a la entidad “Country” (Busque el código “PER” usando find) y en base al resultado imprima el nombre de las ciudades peruanas con población > 700k. **Deberá usar una función lambda** para discriminar el resultado.

```

public class JPAFind {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("world");
        EntityManager em = emf.createEntityManager();

        Country country = em.find(Country.class, "PER");

        if (country != null) {
            List<City> cities = country.getCities().stream().filter(city -> city.getPopulation() > 700000).collect(Collectors.toList());

            System.out.println("Ciudades peruanas con poblacion > 700k");

            for (City city : cities) {
                System.out.println(city.getName());
            }
        } else {
            System.out.println("El pais con codigo PER no fue encontrado");
        }

        em.close();
        emf.close();
    }
}

```

```

Ciudades peruanas con poblacion > 700k
Lima
Arequipa

```

### Parte 03 Proyecto Spring Data JPA (25%)

- Generar un proyecto con Spring Data JPA desde <https://start.spring.io/> con las siguientes características:
  - **Project:** Maven
  - **Language:** Java
  - **Spring Boot:** 3.3.5
  - **Group:** pe.edu.<codigoEstudiante>
  - **Artifact:** cl1-jpa-data-<apellidoPaterno-primerNombre>
  - **Packaging:** Jar
  - **Java:** 23
  - **Dependencies:**
    - Spring Data JPA
    - MySQL Driver
    - Lombok

The Spring Initializr interface is shown with the following configuration:

- Project:** ☐ Gradle - Groovy, ☐ Gradle - Kotlin, ☒ **Maven**
- Language:** ☒ **Java**, ☐ Kotlin, ☐ Groovy
- Spring Boot:** ☐ 3.4.1 (SNAPSHOT), ☒ **3.4.0**, ☐ 3.3.7 (SNAPSHOT), ☐ 3.3.6
- Project Metadata:**
  - Group:
  - Artifact:
  - Name:
  - Description:
  - Package name:
  - Packaging: ☒ **Jar**, ☐ War
  - Java: ☒ **23**, ☐ 21, ☐ 17
- Dependencies:**
  - Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.
  - Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
  - MySQL Driver** (SQL): MySQL JDBC driver.

Buttons at the bottom: GENERATE (CTRL + G), EXPLORE (CTRL + SPACE), SHARE...

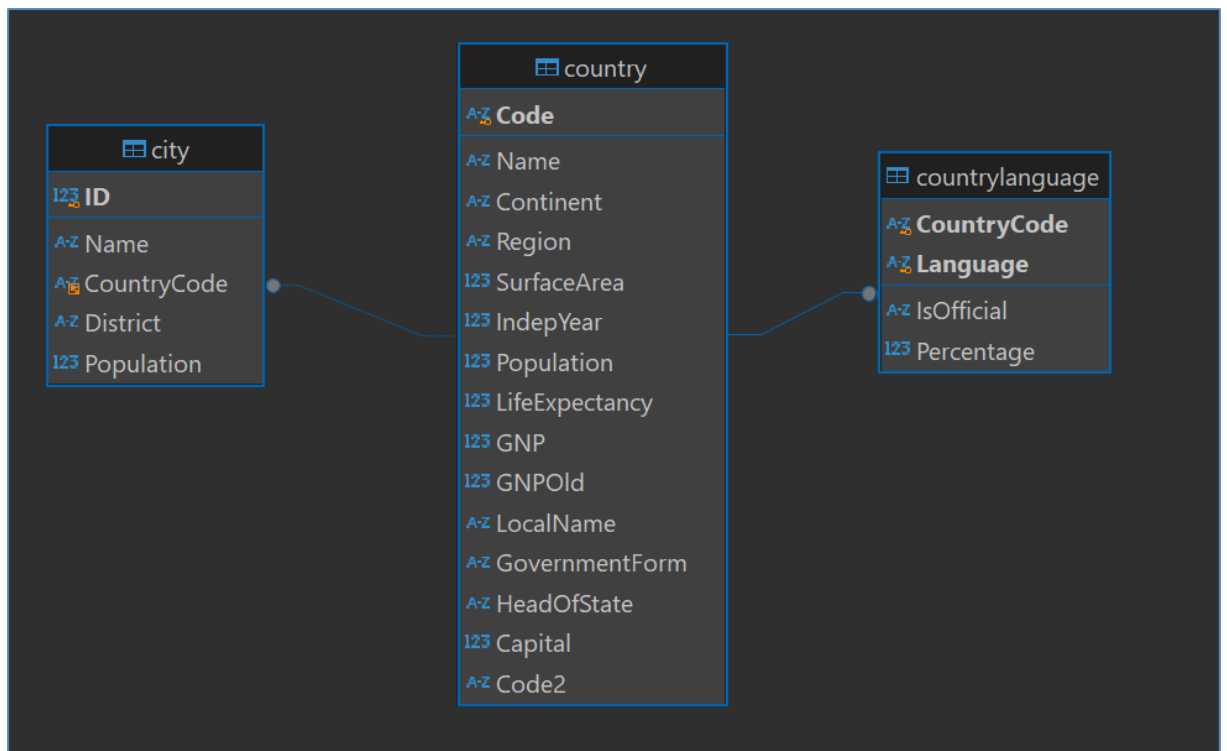
- Configurar el “application.properties” con los datos de conectividad a la bd “world”.

```

1  spring.application.name=cl1-jpa-data-Medina-Lazaro
2  spring.datasource.url=jdbc:mysql://127.0.0.1:3306/world?useSSL=false&serverTimezone=UTC
3  spring.datasource.username=root
4  spring.datasource.password=12345678
5
6  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
7
8  spring.jpa.hibernate.ddl-auto=update
9  spring.jpa.show-sql=true
10 spring.jpa.properties.hibernate.format_sql=true
11 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
12
13 spring.jpa.properties.hibernate.use_sql_comments=true
  
```

- Crear las entidades correspondientes a las siguientes tablas (Las mismas del proyecto anterior):





- Mapear las 3 entidades usando Lombok.
- Definir la estrategia de generación correcta para los PKs.
- Considerar el mapeo de las relaciones de forma bidireccional.

```

package pe.edu.1202221123.cl1_jpa_data_Medina_Lazaro.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity(name = "city")
@Table(name = "city")
@AllArgsConstructor
@NoArgsConstructor
@Data
public class City {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-incremento para ID
    private Integer id; // Clave primaria

    @Column(name = "Name")
    private String name;

    @ManyToOne
    @JoinColumn(name = "CountryCode", referencedColumnName = "Code")
    private Country country; // Relación bidireccional con la entidad Country

    @Column(name = "District")
    private String district;

    @Column(name = "Population")
    private Integer population;
}
  
```

```

package pe.edu.1202221123.cl1_jpa_data_Medina_Lazaro.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity(name = "countrylanguage")
@Table(name = "countrylanguage")
@AllArgsConstructor
@NoArgsConstructor
@Data
public class CountryLanguage {
    @EmbeddedId
    private CountryLanguageId id; // Clave primaria compuesta

    @ManyToOne
    @JoinColumn(name = "CountryCode", referencedColumnName = "Code", insertable = false, updatable = false)
    private Country country; // Relación bidireccional con la entidad Country

    @Column(name = "IsOfficial")
    private String isOfficial;

    @Column(name = "Percentage")
    private Double percentage;
}
  
```



```

City.java Country.java CountryLanguage.java
package pe.edu.i202221123.cl1_jpa_data_Medina_Lazaro.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.Set;

@Entity
@Table(name = "country")
@AllArgsConstructor
@NoArgsConstructor
@Data
public class Country {

    @Id
    @Column(name = "Code")
    private String code; // Clave primaria

    @Column(name = "Name")
    private String name;

    @Column(name = "Continent")
    private String continent;

    @Column(name = "Region")
    private String region;

    @Column(name = "SurfaceArea")
    private Double surfaceArea;

    @Column(name = "IndepYear")
    private Integer independYear;

    @Column(name = "Population")
    private Integer population;

    @Column(name = "LifeExpectancy")
    private Double lifeExpectancy;

    @Column(name = "GNP")
    private Double gnp;

    @Column(name = "GNP0ld")
    private Double gnp0ld;

    @Column(name = "LocalName")
    private String localName;

    @Column(name = "GovernmentForm")
    private String governmentForm;

    @Column(name = "HeadOfState")
    private String headOfState;

    @Column(name = "Capital")
    private Integer capital;

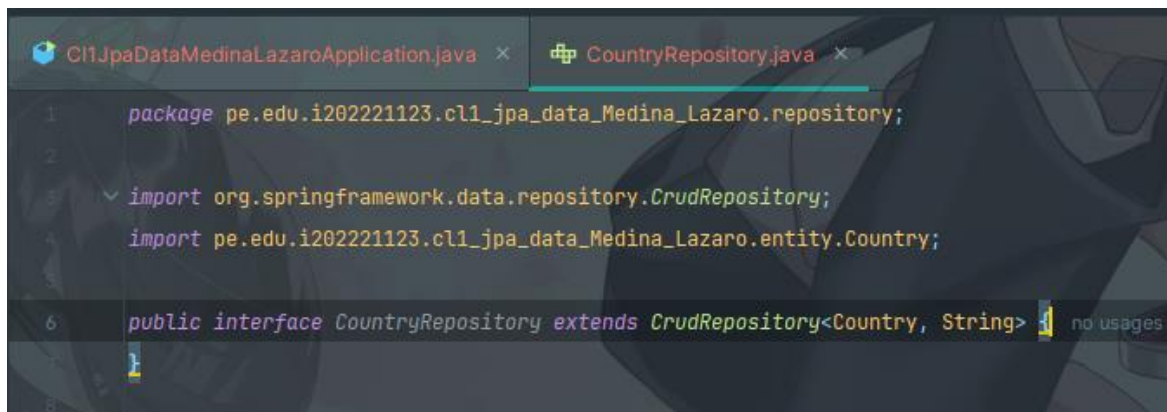
    @Column(name = "Code2")
    private String code2;

    @OneToMany(mappedBy = "country", cascade = CascadeType.ALL)
    private Set<City> cities; // Relación bidireccional con la entidad

    @OneToMany(mappedBy = "country", cascade = CascadeType.ALL)
    private Set<CountryLanguage> countryLanguages;
}

```

- Crear una interfaz “CountryRepository” que extienda de “CrudRepository”.

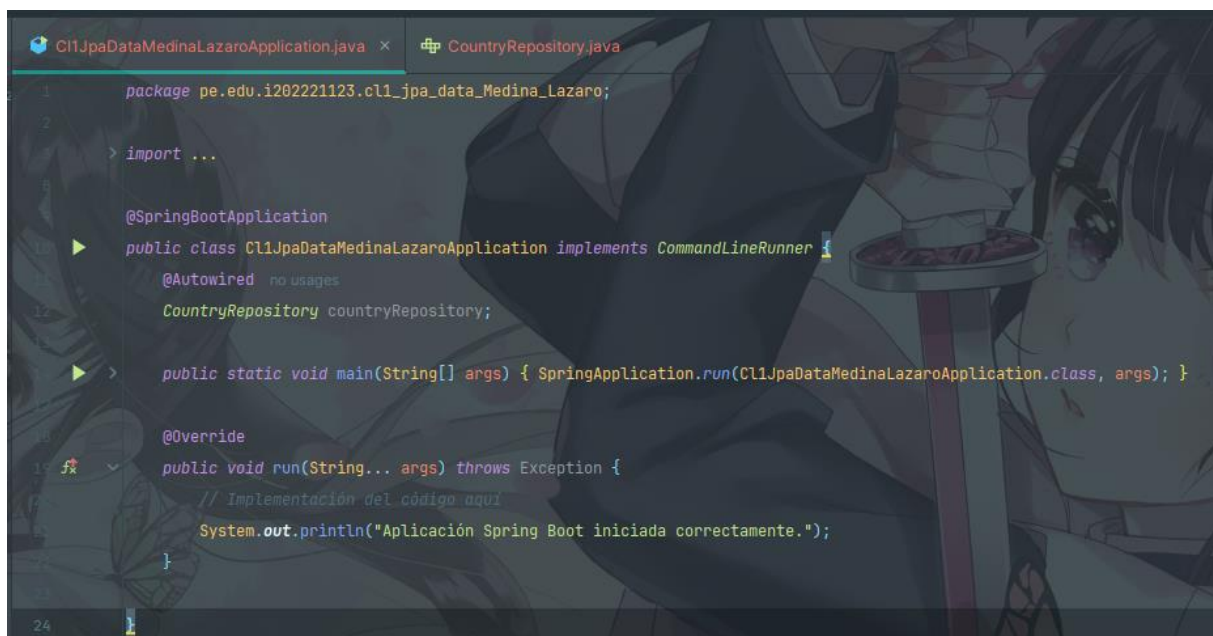


```

1 package pe.edu.i202221123.cl1_jpa_data_Medina_Lazaro.repository;
2
3 import org.springframework.data.repository.CrudRepository;
4 import pe.edu.i202221123.cl1_jpa_data_Medina_Lazaro.entity.Country;
5
6 public interface CountryRepository extends CrudRepository<Country, String> {
7
8 }

```

- Implementar el método “run” definido en la interfaz “CommandLineRunner” desde la clase principal del proyecto de Spring Boot.



```

1 package pe.edu.i202221123.cl1_jpa_data_Medina_Lazaro;
2
3 import ...
4
5 @SpringBootApplication
6 public class Cl1JpaDataMedinaLazaroApplication implements CommandLineRunner {
7     @Autowired
8     CountryRepository countryRepository;
9
10    public static void main(String[] args) { SpringApplication.run(Cl1JpaDataMedinaLazaroApplication.class, args); }
11
12    @Override
13    public void run(String... args) throws Exception {
14        // Implementación del código aquí
15        System.out.println("Aplicación Spring Boot iniciada correctamente.");
16    }
17 }

```

- Deberá implementar las siguientes 3 consultas:
  - **ifPresentOrElse()**  
Imprimir en la terminal los nombres de los lenguajes que se hablan en el país “ARG” (Argentina). En caso de no obtener resultado, deberá imprimir los nombres de los lenguajes del país “PER” (Perú).

```
String countryCodeArg = "ARG";
String countryCodePer = "PER";

countryRepository.findById(countryCodeArg).ifPresentOrElse(
    country -> {
        System.out.println("Lenguajes de Argentina (ARG):");
        country.getCountryLanguages().forEach(language ->
            System.out.println(language.getId().getLanguage()));
    },
    () -> {
        System.out.println("No se encontraron lenguajes para Argentina. Mostrando lenguajes de Perú (PER):");
        countryRepository.findById(countryCodePer).ifPresent(country -> {
            country.getCountryLanguages().forEach(lenguaje ->
                System.out.println(lenguaje.getId().getLanguage()));
        });
    }
);
```

- **deleteAllById()**

Eliminar 2 países: "COL" y "ARG". La eliminación deberá ser cascada y borrará sus ciudades y lenguajes correspondientes.

```
List<String> CountryCodesToDelete = Arrays.asList("COL", "ARG");
countryRepository.deleteAllById(CountryCodesToDelete);
System.out.println("Eliminados son: " + CountryCodesToDelete);
```

- Volver a ejecutar la primera consulta, pues al eliminar "ARG", deberá ejecutarse el flujo alterno. (Deberá restaurar la BD desde la terminal en cada prueba)

```

String countryCodeArg = "ARG";
String countryCodePer = "PER";

countryRepository.findById(countryCodeArg).ifPresentOrElse(
    country -> {
        System.out.println("Lenguajes de Argentina (ARG):");
        country.getCountryLanguages().forEach(language ->
            System.out.println(language.getId().getLanguage())
        );
    },
    () -> {
        System.out.println("No se encontraron lenguajes para Argentina. Mostrando lenguajes de Perú (PER):");
        countryRepository.findById(countryCodePer).ifPresent(country -> {
            country.getCountryLanguages().forEach(language ->
                System.out.println(language.getId().getLanguage())
            );
        });
    }
);

// Eliminar países: COL y ARG
List<String> countryCodesToDelete = Arrays.asList("COL", "ARG");
countryRepository.deleteAllById(countryCodesToDelete);
System.out.println("Países eliminados: " + countryCodesToDelete);

// Repetir consulta para activar flujo alternativo
countryRepository.findById(countryCodeArg).ifPresentOrElse(
    country -> {
        System.out.println("Lenguajes de Argentina (ARG):");
        country.getCountryLanguages().forEach(language ->
            System.out.println(language.getId().getLanguage())
        );
    },
    () -> {
        System.out.println("No se encontraron lenguajes para Argentina. Mostrando lenguajes de Perú (PER):");
        countryRepository.findById(countryCodePer).ifPresent(country -> {
            country.getCountryLanguages().forEach(language ->
                System.out.println(language.getId().getLanguage())
            );
        });
    }
);

```

#### Parte 04 Gestión de conexiones (25%)

- Crear una clase de configuración denominada “ConexionesConfig.java”, en ella deberá implementar una personalización del DataSource de HikariCP. Considere los siguientes valores para el pool de conexiones:
  - MaxmimunPoolSize: 30
  - MinimumIdle: 4
  - IdleTimeout: 4 minutos

- ConnectionTimeout: 45 segundos

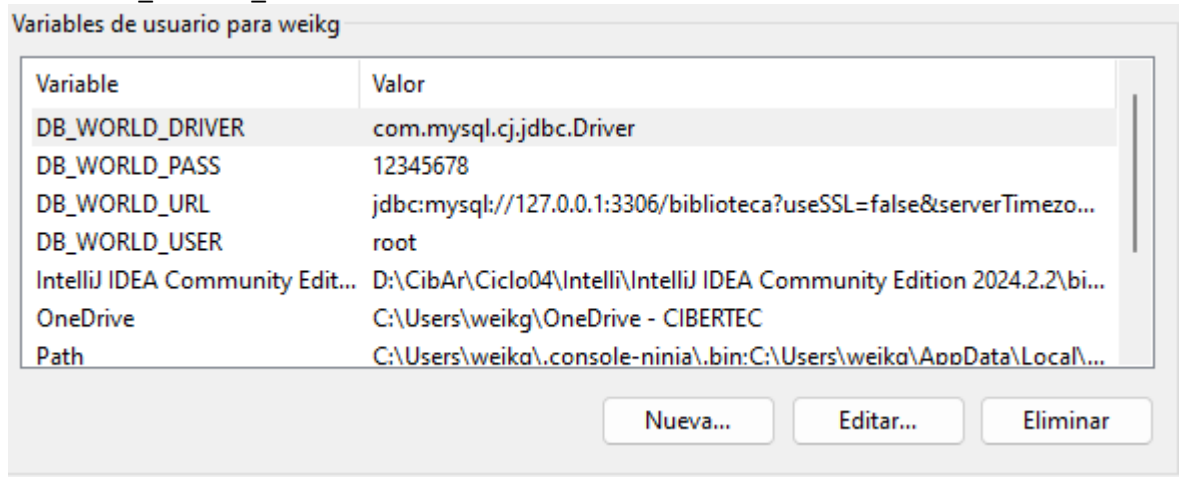


```
Cl1JpaDataMedinaLazaroApplication.java x ConexionesConfig.java x Cou

1 package pe.edu.i202221123.cl1_jpa_data_Medina_Lazaro.config;
2
3 > import ...
4
5
6
7
8
9 @Configuration no usages
10 public class ConexionesConfig {
11     @Value("${DB_SAKILA_URL}") 1 usage
12     private String dbSakilaUrl;
13
14     @Value("${DB_SAKILA_USER}") 1 usage
15     private String dbSakilaUser;
16
17     @Value("${DB_SAKILA_PASS}") 1 usage
18     private String dbSakilaPass;
19
20     @Value("${DB_SAKILA_DRIVER}") 1 usage
21     private String dbSakilaDriver;
22
23
24
25 @Bean no usages
26 public HikariDataSource hikariDataSource() {
27     HikariConfig hc = new HikariConfig();
28
29
30     hc.setJdbcUrl(dbSakilaUrl);
31     hc.setUsername(dbSakilaUser);
32     hc.setPassword(dbSakilaPass);
33     hc.setDriverClassName(dbSakilaDriver);
34
35     hc.setMaximumPoolSize(30);
36     hc.setMinimumIdle(4);
37     hc.setIdleTimeout(240000);
38     hc.setConnectionTimeout(45000);
39
40     System.out.println("##### HikariCP initialized #####");
41     return new HikariDataSource(hc);
42 }
43 }
```



- Las credenciales del DataSource (Parámetros de conexión a la BD) no deben ser visibles en el código fuente. Para ello deberá crear las siguientes variables de entorno:
  - DB\_WORLD\_URL
  - DB\_WORLD\_USER
  - DB\_WORLD\_PASS
  - DB\_WORLD\_DRIVER



- Luego de configurar el DataSource, ¿Es necesario proporcionar las credenciales desde el archivo “application.properties”? ¿Por qué? Explique con sus propias palabras.

No porque ya lo tenemos en el DataSource y como no debe ser visible entonces se quita de application, no tendría caso tenerlo ahí.

- ¿Por qué debo o no, configurar un JNDI en Spring Boot?  
Es necesario cuando trabajamos en un entorno de servidor de aplicaciones o contenedor de aplicaciones empresarial y no es necesario cuando estamos trabajando en desarrollo local o con aplicaciones autónomas.