

CURSO : Desarrollo de Aplicaciones Web I (0265)
PROFESOR : César Enrique Santos Torres
CICLO : Quinto
SECCIÓN : LX2033X9705
GRUPO : 2024335545
FECHA : 24/11/2024 23:59hrs
DURACIÓN : 3 horas

NOTA

ALUMNO (A) : BRENDA LUZ ALEJANDRO BECERRA

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.

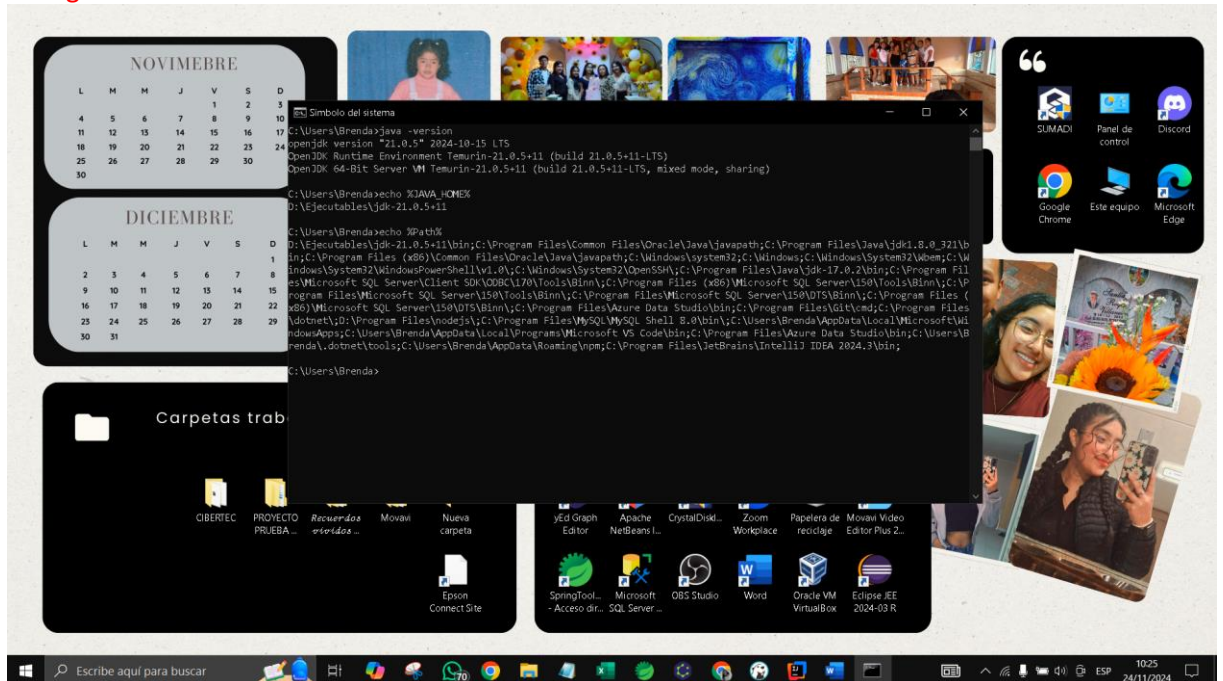
CONSOLIDADO

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%)

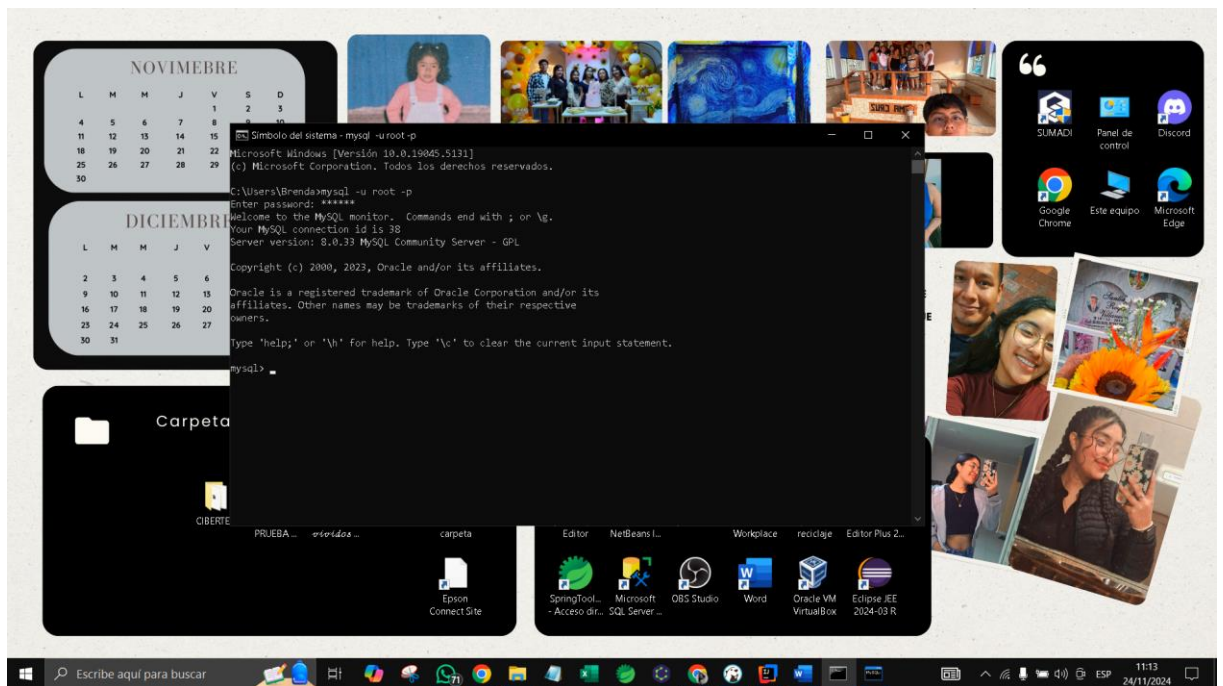
- Descargar JDK versión 23 de <https://adoptium.net/es/temurin/releases/>
- Configurar variable de entorno JAVA_HOME y Path
- Validar configuración Java con los siguientes comandos (**Use cmd**):
 - java -version
 - echo %JAVA_HOME%
 - echo %Path%

<images>



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

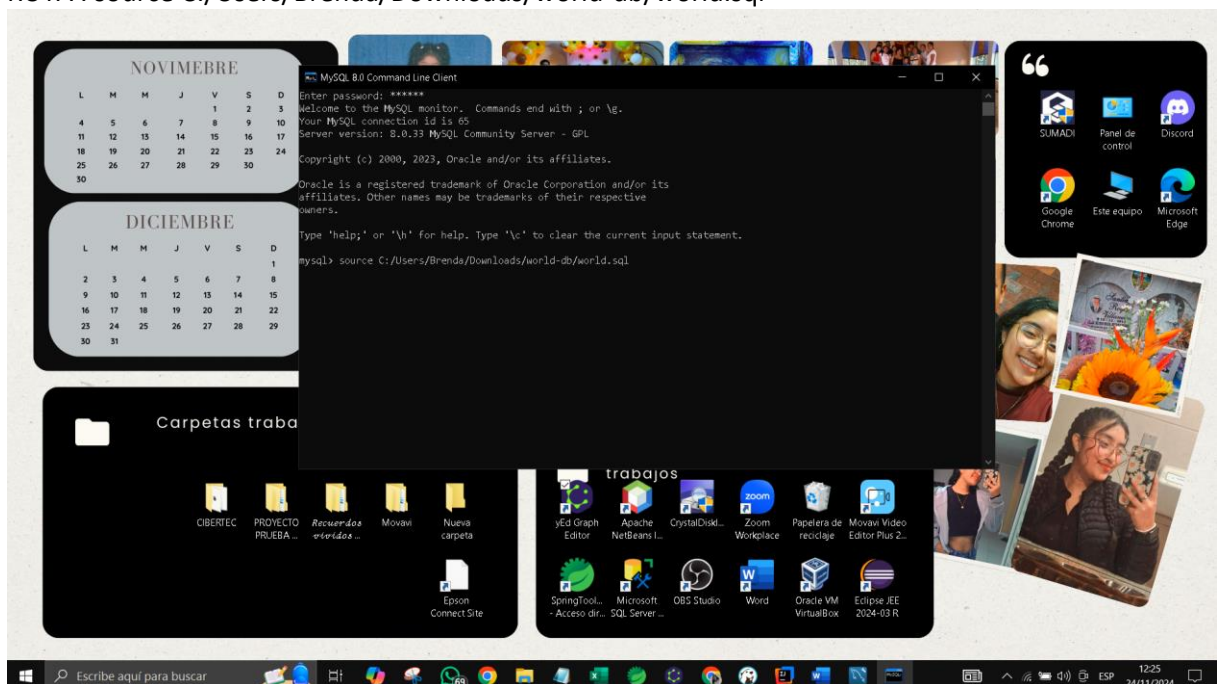
<images>



- Restaurar bd “world” de <https://downloads.mysql.com/docs/world-db.zip> (Use cmd):
 - Use el comando: `source <ruta-archivo-world.sql>;`

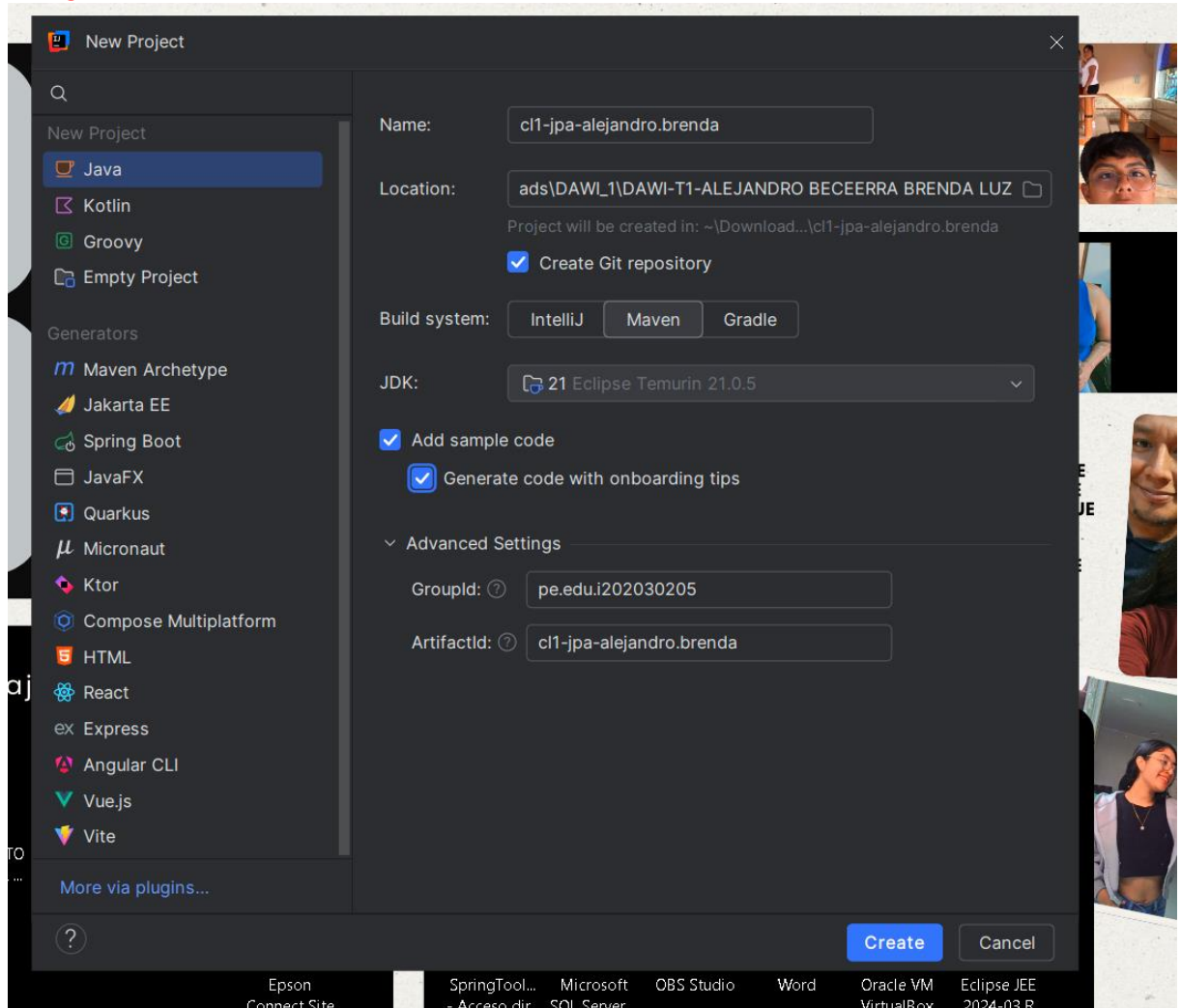
<images>

ruta : `source C:/Users/Brenda/Downloads/world-db/world.sql`



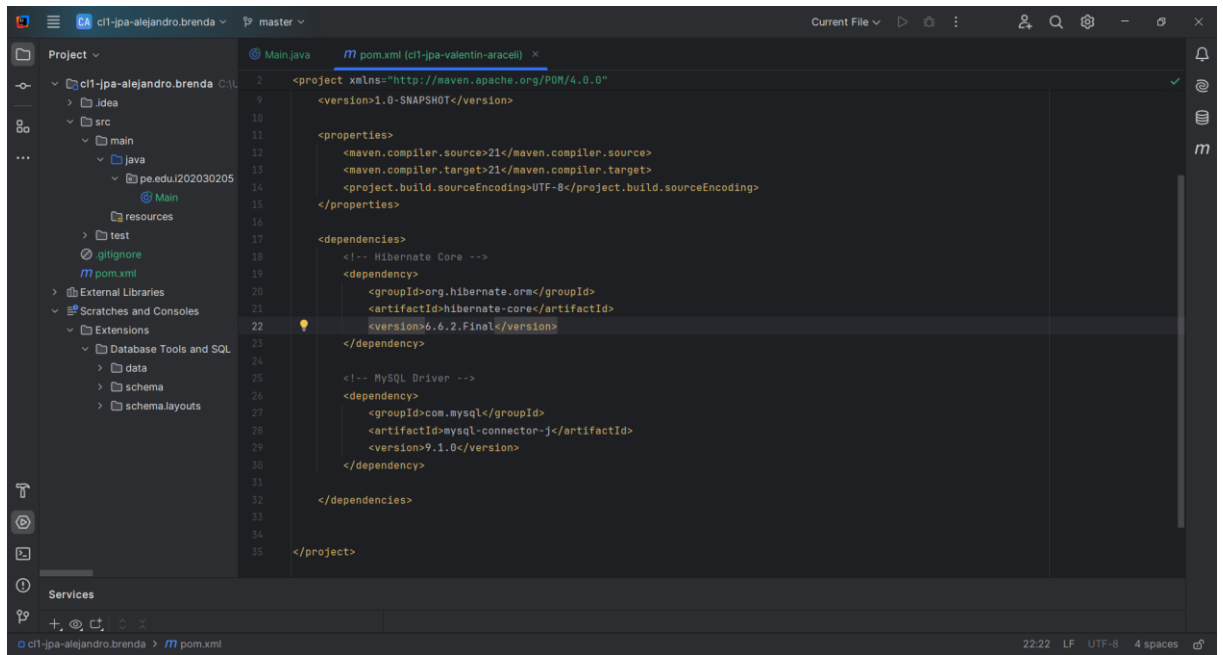
- **Build system:** Maven
- **JDK:** Eclipse Temurin-23
- **Advanced Settings:**
 - **GroupId:** pe.edu.<codigoEstudiante>
 - **Artifact:** cl1-jpa-<apellidoPaterno-primerNombre>

<images>



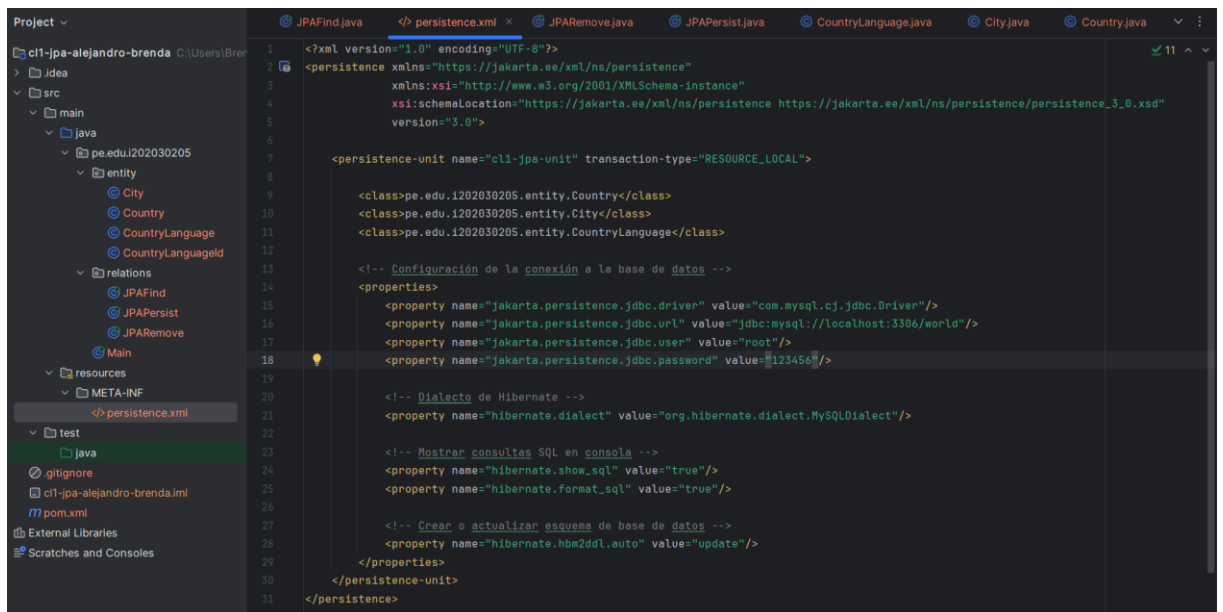
- Configurar dependencias en el pom.xml
 - Hibernate (6.6.2.Final)
 - Driver de MySQL (9.1.0)

<images>

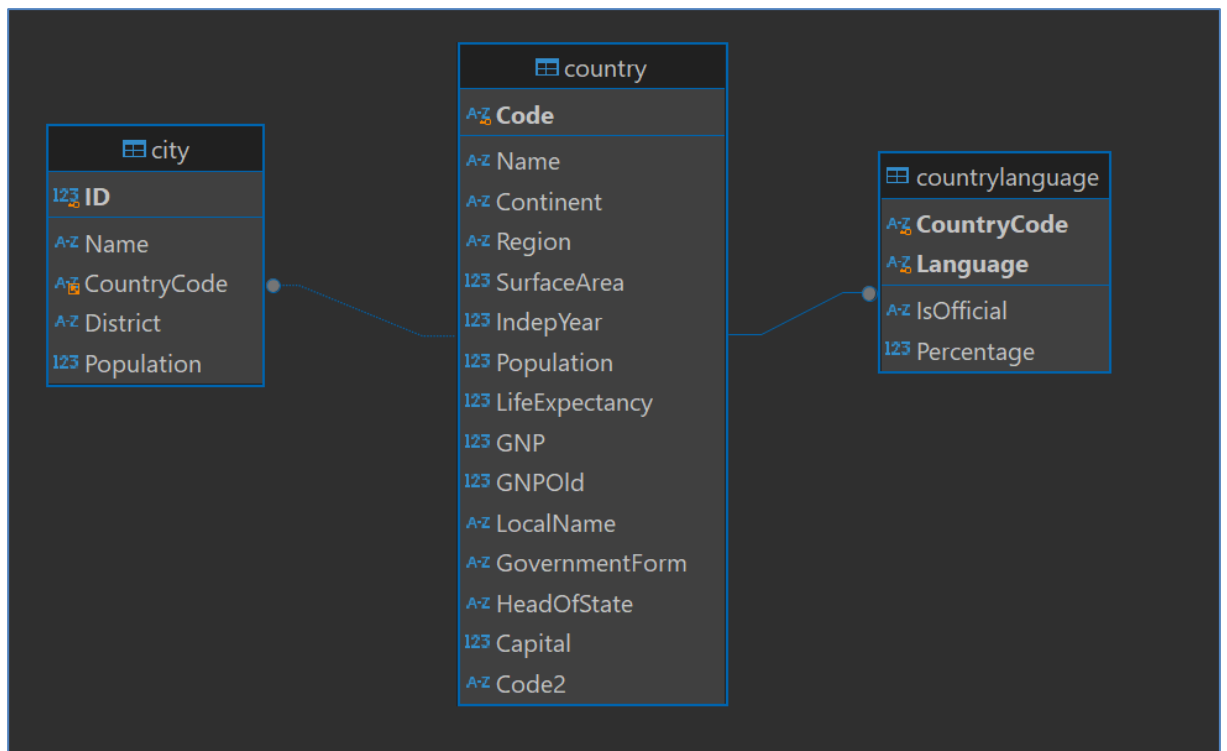


- Configurar unidad de persistencia en archivo “persistence.xml”

<images>



- 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.

<images>

```

package pe.edu.1202030205.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "city")
public class City {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "name") 2 usages
    private String name;

    @Column(name = "district") 2 usages
    private String district;

    @Column(name = "population") 2 usages
    private int population;

    // Relación con Country
    @ManyToOne 2 usages
    @JoinColumn(name = "CountryCode", referencedColumnName = "Code")
    private Country country;

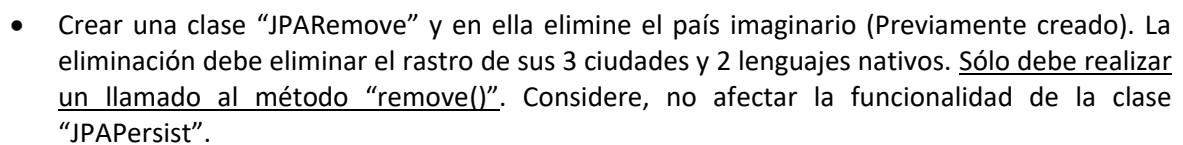
    // Getters y Setters
    public int getId() { return id; } no usages
    public void setId(int id) { this.id = id; } no usages
    public String getName() { return name; } no usages
    public void setName(String name) { this.name = name; } no usages
    public String getDistrict() { return district; } no usages
}
  
```

```
1 package pe.edu.i202030205.entity;
2
3 import jakarta.persistence.*;
4
5 import java.util.List;
6
7
8 @Entity
9 @Table(name = "country")
10
11 public class Country {
12     @Id
13     @Column(name = "Code")
14     private String code;
15
16     @Column(name = "Name") 3 usages
17     private String name;
18
19     @Column(name = "Continent") 3 usages
20     private String continent;
21
22     @Column(name = "Region") 3 usages
23     private String region;
24
25     @Column(name = "SurfaceArea") 3 usages
26     private double surfaceArea;
27
28     @Column(name = "IndepYear") 3 usages
29     private Integer indepYear;
30 }
```

```
1 package pe.edu.i202030205.entity;
2
3 import jakarta.persistence.*;
4
5 @Entity
6 @Table(name = "countrylanguage")
7
8 public class CountryLanguage {
9     @EmbeddedId 2 usages
10     private CountryLanguageId id;
11
12     @Column(name = "IsOfficial") 2 usages
13     private boolean isOfficial;
14
15     @Column(name = "Percentage") 2 usages
16     private double percentage;
17
18     // Relación con Country
19     @ManyToOne 2 usages
20     @MapsId("countryCode")
21     @JoinColumn(name = "CountryCode", referencedColumnName = "Code")
22     private Country country;
23
24     // Getters y Setters
25     public CountryLanguageId getId() { return id; } no usages
26     public void setId(CountryLanguageId id) { this.id = id; } no usages
27     public boolean isOfficial() { return isOfficial; } no usages
28     public void setOfficial(boolean official) { isOfficial = official; } no usages
29     public double getPercentage() { return percentage; } no usages
30     public void setPercentage(double percentage) { this.percentage = percentage; } no usages
31     public Country getCountry() { return country; } no usages
32     public void setCountry(Country country) { this.country = country; } no usages
33 }
```

```
1 package pe.edu.i202030205.entity;
2
3 import jakarta.persistence.Embeddable;
4 import java.io.Serializable;
5 import java.util.Objects;
6
7 @Embeddable 5 usages
8 public class CountryLanguageId implements Serializable {
9     private String countryCode; 5 usages
10     private String language; 5 usages
11
12     // Getters y Setters
13
14     public String getCountryCode() { return countryCode; } no usages
15     public void setCountryCode(String countryCode) { this.countryCode = countryCode; } no usages
16     public String getLanguage() { return language; } no usages
17     public void setLanguage(String language) { this.language = language; } no usages
18
19     // equals() y hashCode()
20     @Override
21     public boolean equals(Object o) {
22         if (this == o) return true;
23         if (o == null || getClass() != o.getClass()) return false;
24         CountryLanguageId that = (CountryLanguageId) o;
25         return Objects.equals(countryCode, that.countryCode) &&
26             Objects.equals(language, that.language);
27     }
28
29     @Override
30     public int hashCode() {
31         return Objects.hash(countryCode, language);
32     }
33 }
```


- <images>

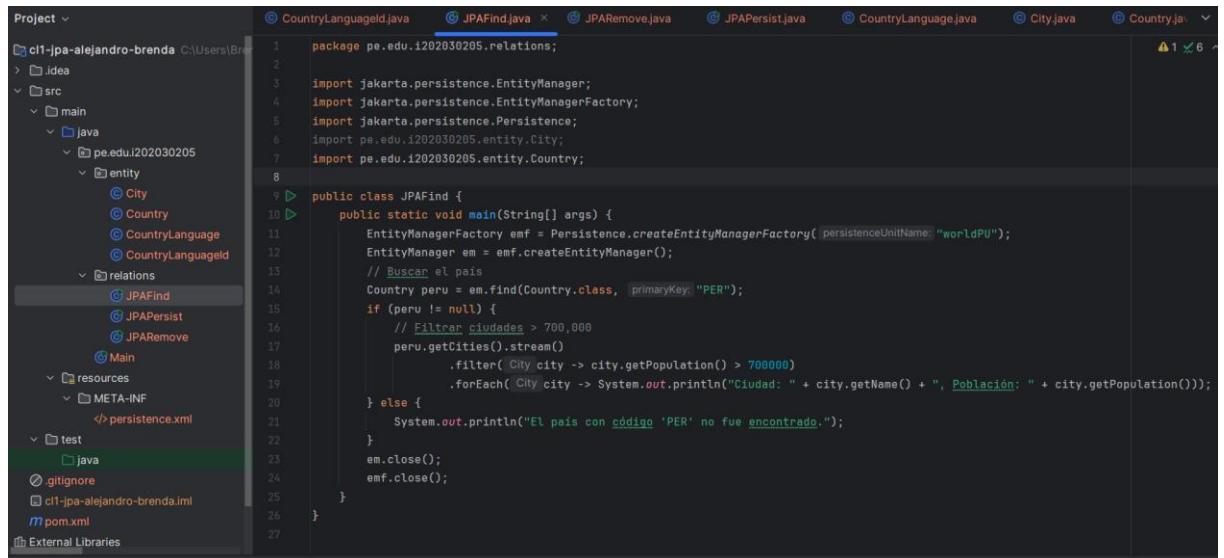


<images>



- 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.

<images>



```

1 package pe.edu.i202030205.relations;
2
3 import jakarta.persistence.EntityManager;
4 import jakarta.persistence.EntityManagerFactory;
5 import jakarta.persistence.Persistence;
6 import pe.edu.i202030205.entity.City;
7 import pe.edu.i202030205.entity.Country;
8
9 public class JPAFind {
10     public static void main(String[] args) {
11         EntityManagerFactory emf = Persistence.createEntityManagerFactory("worldPU");
12         EntityManager em = emf.createEntityManager();
13         // Buscar el país
14         Country peru = em.find(Country.class, primaryKey: "PER");
15         if (peru != null) {
16             // Filtrar ciudades > 700,000
17             peru.getCities().stream()
18                 .filter(City city -> city.getPopulation() > 700000)
19                 .forEach(City city -> System.out.println("Ciudad: " + city.getName() + ", Población: " + city.getPopulation()));
20         } else {
21             System.out.println("El país con código 'PER' no fue encontrado.");
22         }
23         em.close();
24         emf.close();
25     }
26 }
27

```

Parte 03 Proyecto Spring Data JPA (25%)

- Git : <https://github.com/BrendaAlejandro/DAWI-1-Pregunta03-04.git>
- 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

<images>

Project

☐ Gradle - Groovy
☐ Gradle - Kotlin
☒ Java
☐ Kotlin
☐ Groovy

☒ Maven

Spring Boot

☐ 3.4.1 (SNAPSHOT)
☒ 3.4.0
☐ 3.3.7 (SNAPSHOT)
☐ 3.3.6

Project Metadata

Group

pe.edu.i202030205

Artifact

cl1-jpa-data-alejandro-brenda

Name

cl1-jpa-data-alejandro-brenda

Description

Demo project for Spring Boot

Package name

pe.edu.i202030205.cl1-jpa-data-alejandro-brenda

Packaging

☒ Jar
☐ War

Java

☒ 23
☐ 21
☐ 17

Dependencies

ADD DEPENDENCIES... CTRL + B

Lombok

DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

MySQL Driver

SQL

MySQL JDBC driver.

Spring Data JPA

SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

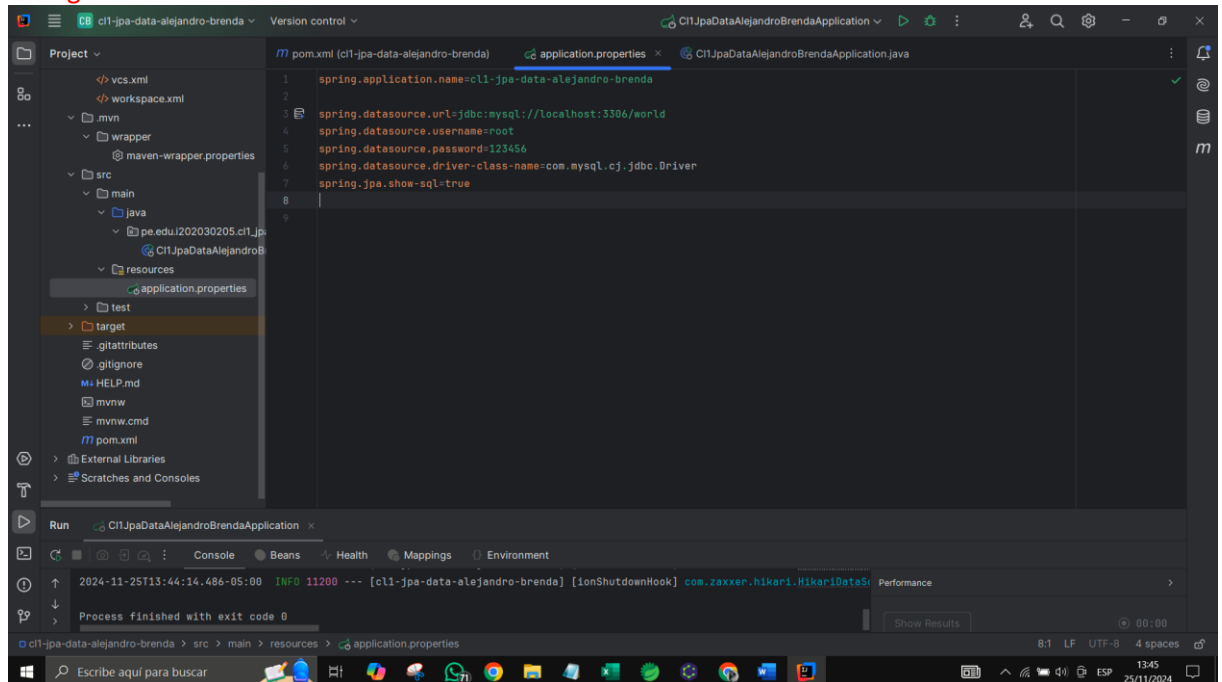
GENERATE CTRL + G

EXPLORE CTRL + SPACE

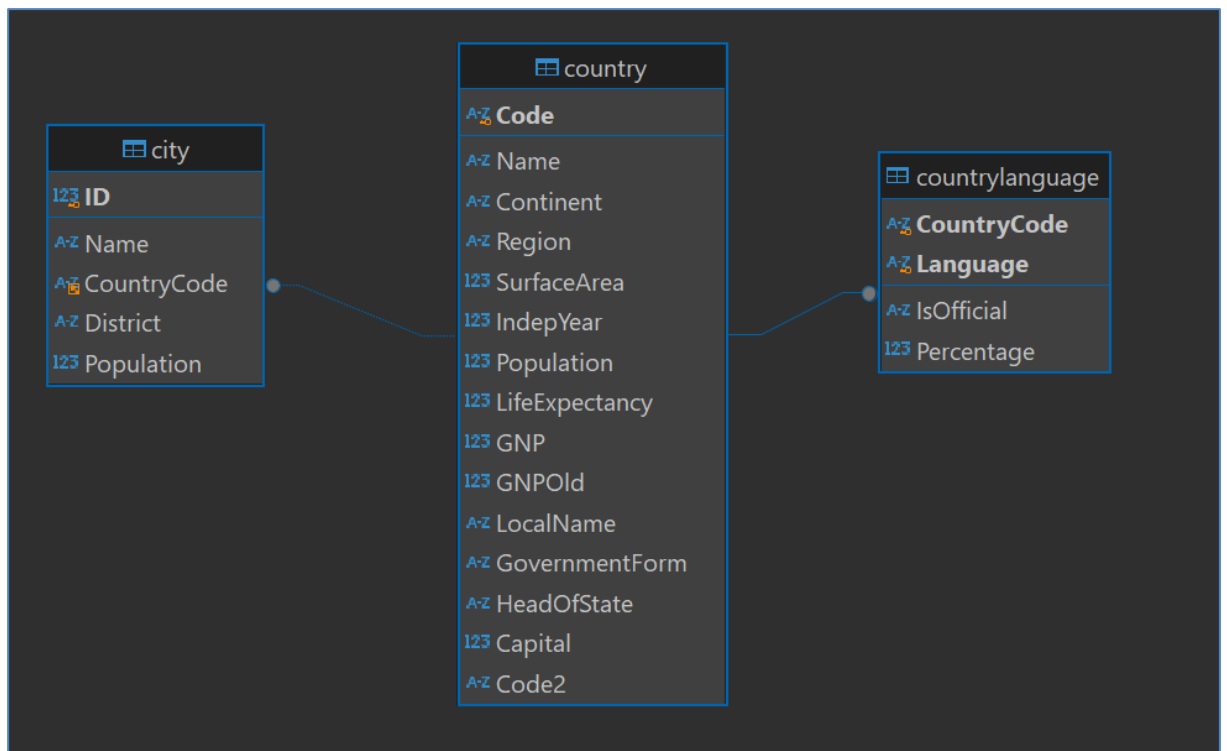
SHARE...

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

<images>



- 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.

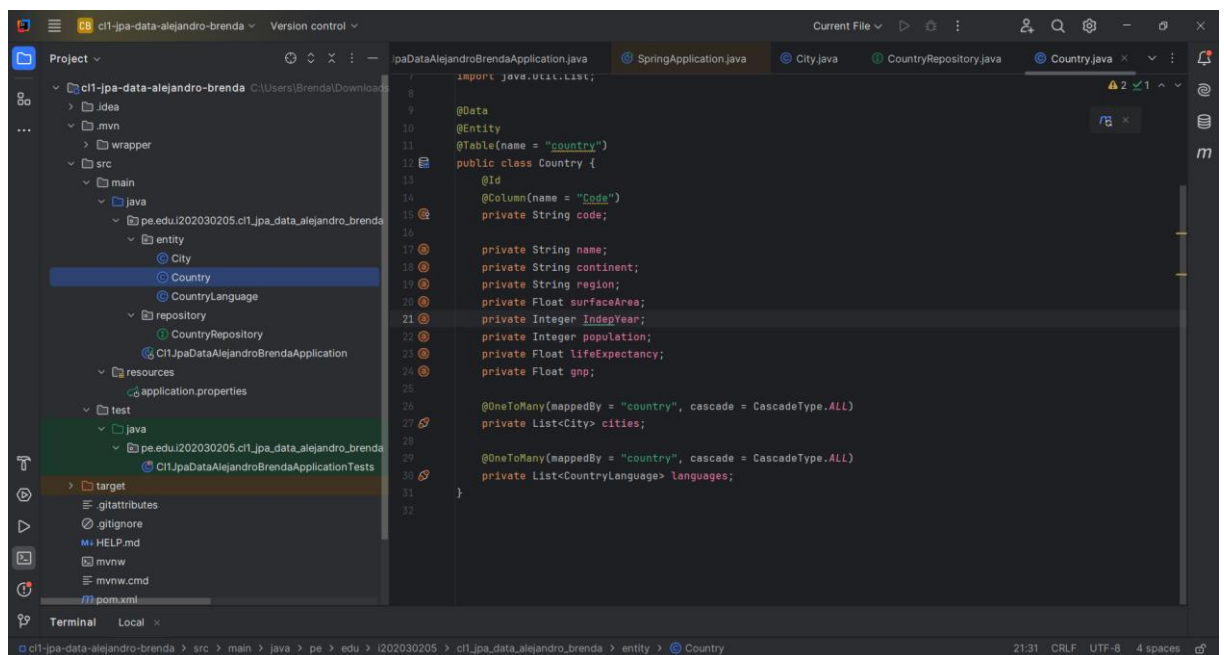
<images>

- City

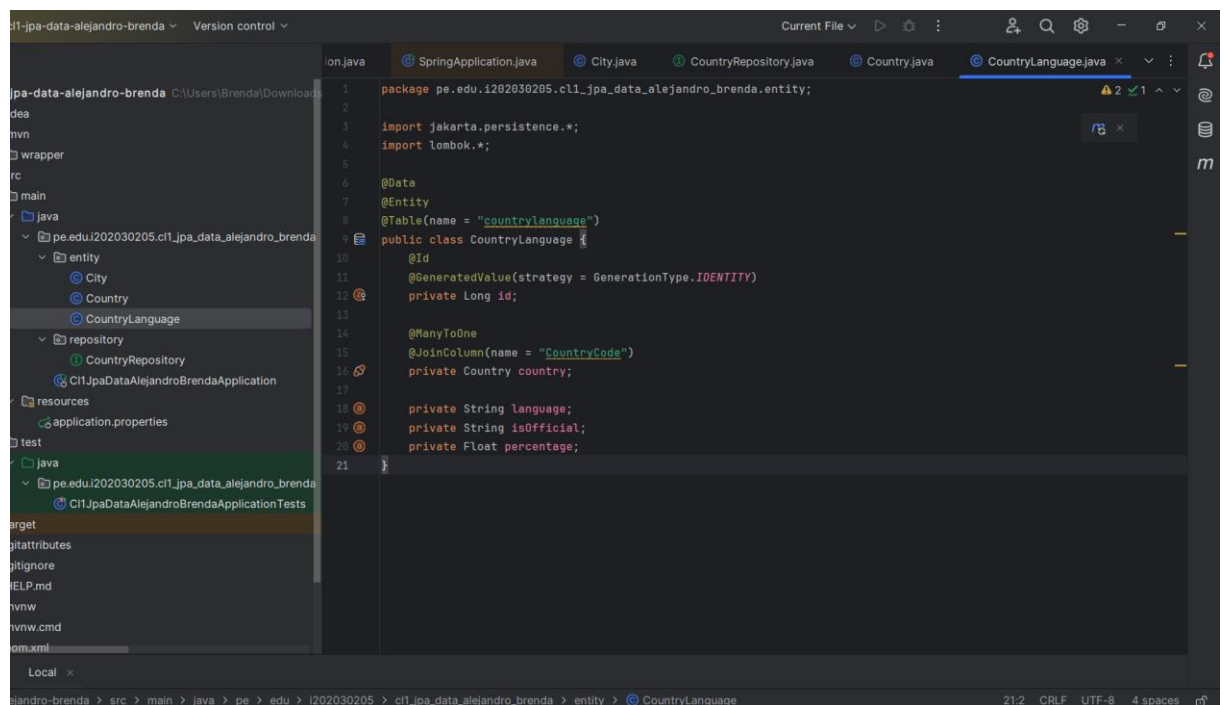
```

1 package pe.edu.i202030205.cl1_jpa_data_alejandro_brenda.entity;
2
3 import jakarta.persistence.*;
4 import lombok.Data;
5
6
7 @Data
8 @Entity
9 @Table(name = "city")
10 public class City {
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private Integer id;
14
15     private String name;
16
17     @ManyToOne
18     @JoinColumn(name = "CountryCode")
19     private Country country;
20
21     private String district;
22     private Integer population;
23 }
  
```

- Country

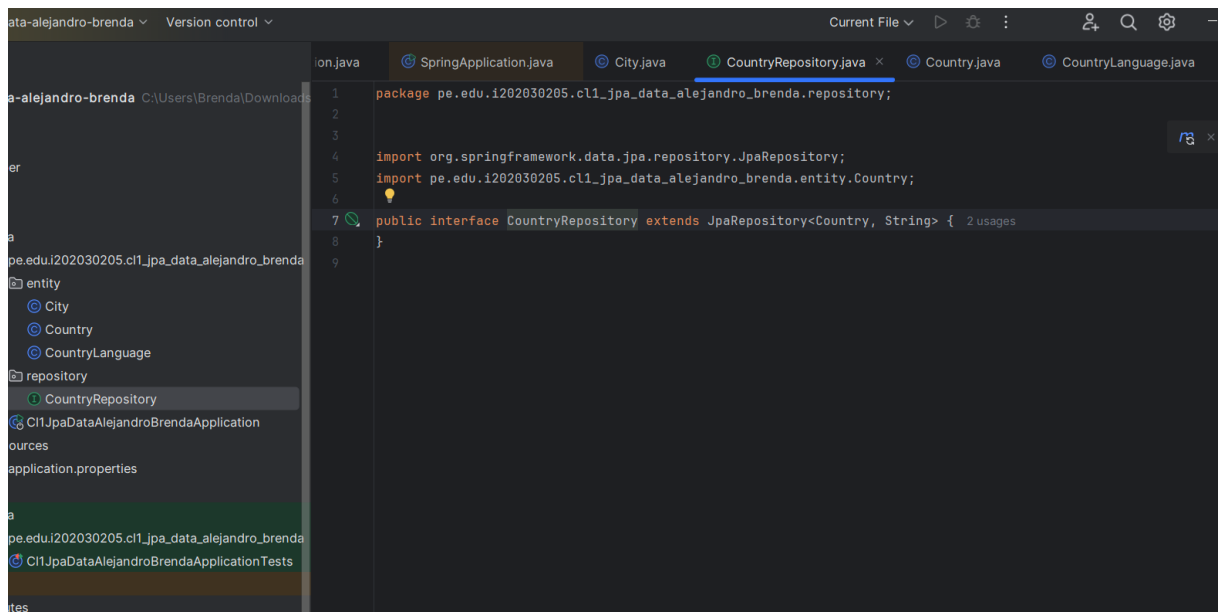


- countrylanguage



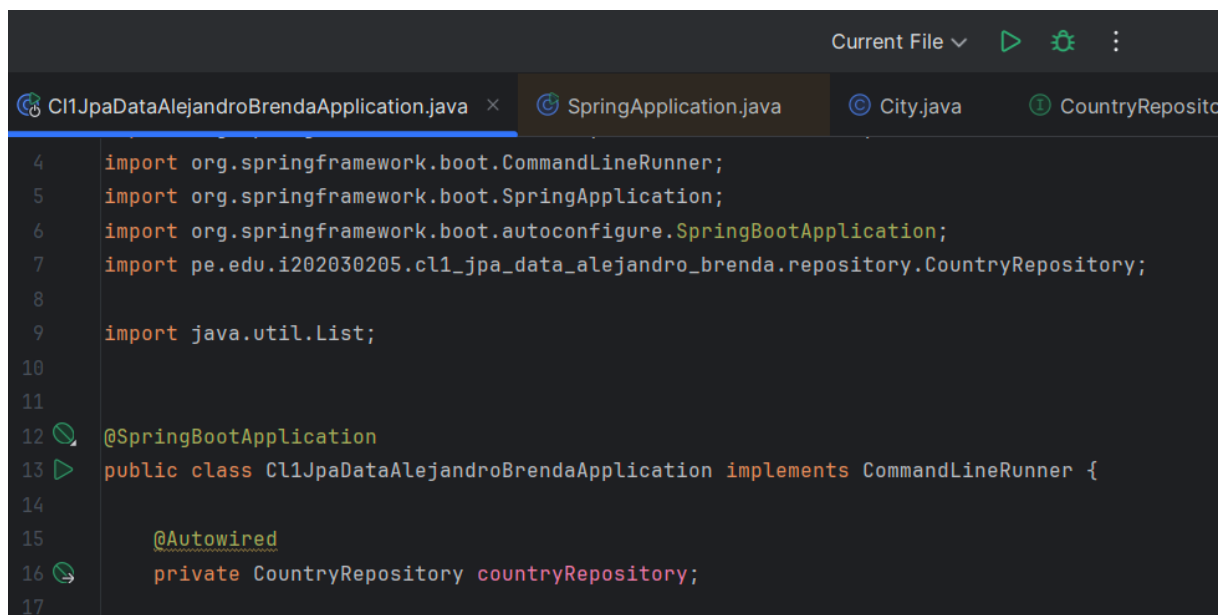
- Crear una interfaz "CountryRepository" que extienda de "CrudRepository".

<images>



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

<images>



- Deberá implementar las siguientes 2 consultas:
 - **Use “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).

<images>

```
@Override
public void run(String... args) throws Exception {
    // Consulta con ifPresentOrElse
    countryRepository.findById("ARG").ifPresentOrElse(
        Country country -> country.getLanguages().forEach( CountryLanguage lang ->
            System.out.println("Language: " + lang.getLanguage())) ,
        () -> countryRepository.findById("PER").ifPresent( Country peru ->
            peru.getLanguages().forEach( CountryLanguage lang ->
                System.out.println("Language: " + lang.getLanguage())) )
    );
}
```

- Use “deleteAllById()”

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

<images>

```
// Eliminar países con deleteAllById
countryRepository.deleteAllById(List.of("COL", "ARG"));
```

Nota:

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)

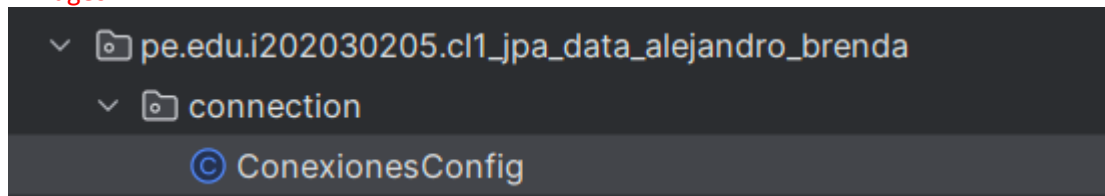
Parte 04 Gestión de conexiones (25%)

- Git: <https://github.com/BrendaAlejandro/DAWI-1-Pregunta03-04.git>

- 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:

- MaximunPoolSize: 30
- MinimumIdle: 4
- IdleTimeout: 4 minutos
- ConnectionTimeout: 45 segundos

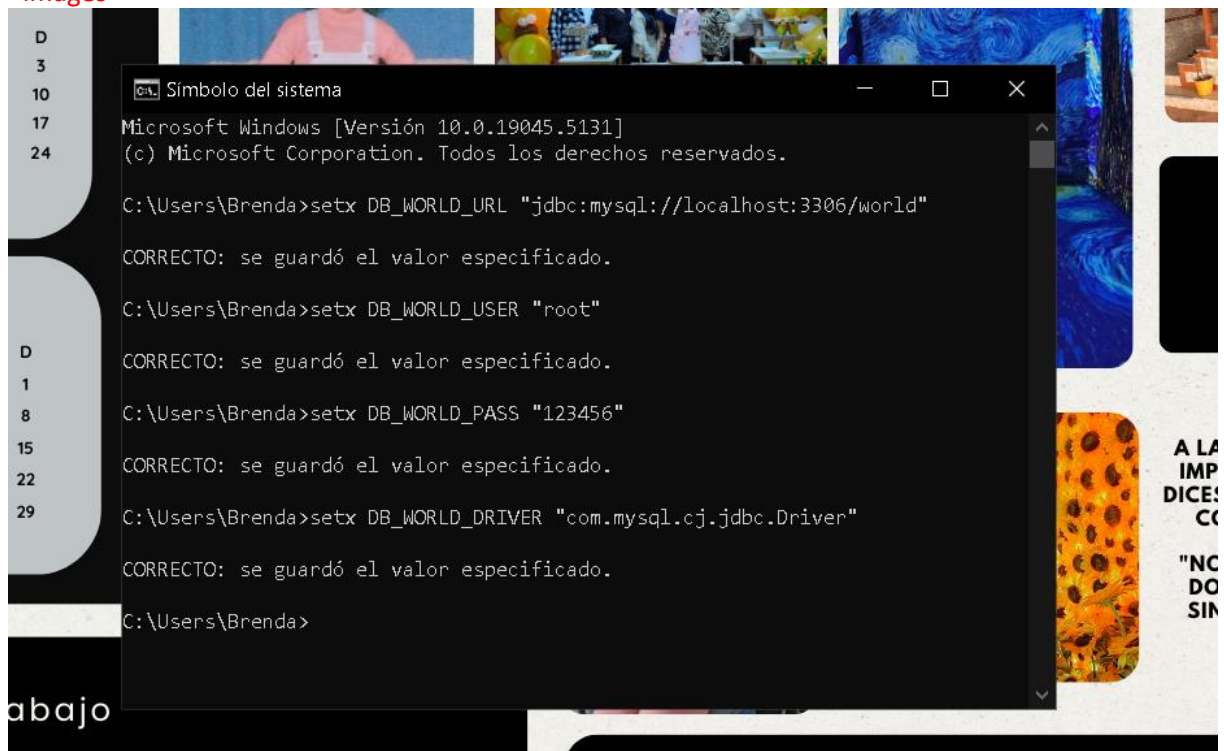
<images>



```
ConexionesConfig.java x SpringApplication.java City.java CountryRepository.java Country.java CountryLanguage.java
1 package pe.edu.1202030205.cl1_jpa_data_alejandra_brenda.connection;
2
3 import com.zaxxer.hikari.HikariConfig;
4 import com.zaxxer.hikari.HikariDataSource;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7
8 import javax.sql.DataSource;
9
10 @Configuration
11 public class ConexionesConfig {
12
13     @Bean
14     public DataSource dataSource() {
15         HikariConfig config = new HikariConfig();
16
17         // Configuración del pool
18         config.setMaximumPoolSize(30);
19         config.setMinimumIdle(4);
20         config.setIdleTimeout(240000);
21         config.setConnectionTimeout(45000);
22
23         // Configuración de la base de datos
24         config.setJdbcUrl("jdbc:mysql://localhost:3306/tu_base_de_datos"); // Cambiar URL
25         config.setUsername("tu_usuario"); // Cambiar usuario
26         config.setPassword("tu_contraseña"); // Cambiar contraseña
27         config.setDriverClassName("com.mysql.cj.jdbc.Driver");
28
29         return new HikariDataSource(config);
30     }
31 }
```

- 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

<images>



- Luego de configurar el DataSource, **¿Es necesario proporcionar las credenciales desde el archivo "application.properties"? ¿Por qué?**

No, no necesito proporcionar las credenciales en el archivo application.properties si ya las configuré mediante variables de entorno. Utilice variables de entorno para almacenar credenciales (como URL de bases de datos, usuarios, contraseñas y controladores) que no están expuestas en el código o los archivos de configuración.

Esto tiene muchas ventajas. En primer lugar, mejora la seguridad porque las credenciales no son visibles para otros desarrolladores y no se almacenan en el código. En segundo lugar, proporciona flexibilidad porque puedo cambiar las credenciales según el entorno (desarrollo, prueba, producción) simplemente configurando variables de entorno sin cambiar el archivo application.properties.

Por lo tanto, no es necesario leer las credenciales de las variables de entorno, deben incluirse en el archivo de la aplicación .properties, lo que hace que la aplicación sea más segura y más fácil de administrar en diferentes entornos.

- **¿Por qué debo o no, configurar un JNDI en Spring Boot?**

No es necesario configurar JNDI en Spring Boot, pero dependiendo de cómo esté configurada su aplicación, puede resultar útil.

- Razón para configurar JNDI:

1. Entorno del servidor de aplicaciones: si la aplicación se ejecuta en un servidor como Tomcat, JBoss o WebLogic, estos servidores generalmente usan JNDI para administrar los recursos de conexión de la base de datos. En este caso, la configuración JNDI hace que el servidor se encargue de gestionar la conexión, y la aplicación sólo obtiene recursos de allí.
2. Centralización de recursos: el uso de JNDI permite que el servidor administre de forma centralizada las conexiones desde la base de datos, lo cual es muy útil cuando hay varias conexiones de bases de datos. Aplicaciones que necesitan acceder a los mismos datos de la base de datos. Esto simplifica la instalación y el mantenimiento, especialmente en entornos de producción.

- Razón para no configurar JNDI:

1. Sin servidor de aplicaciones: no se requiere configuración a menos que esté utilizando un servidor de aplicaciones que utilice JNDI para administrar recursos. En su lugar, puedes configurar la fuente de datos directamente usando una herramienta como HikariCP. Lo que sea más fácil.
2. Simplicidad y control: si no necesita la complejidad de JNDI, es más fácil configurarlo directamente en su aplicación. Además, tiene más control sobre cómo se manejan las conexiones de la base de datos.