

ADR- Creación Dinámica de Entidades JPA

CONFIGURA

Exported on 02/29/2024

Table of Contents

1 ADR - Configura - 0000013

You are using an UNLICENSED copy of **Scroll PDF Exporter for Confluence**. Do you find Scroll PDF Exporter useful? Consider purchasing it today:
<https://marketplace.atlassian.com/apps/7019/scroll-pdf-exporter-for-confluence?tab=overview&hosting=datacenter>

1 ADR - Configura - 000001

Estado	EN VIGOR
Partes interesadas	DESARROLLO SOFTWARE ECONOMICO-FINANCIERO (NTTDATA) ¹ AREA GOBERNANZA ²
TAGS	
Tomada el	29/02/2024
Responsable	

1 https://ws001.sspa.juntadeandalucia.es/confluence/display/~SC00_FEF
2 https://ws001.sspa.juntadeandalucia.es/confluence/display/~SRVC_GOBERNANZA

Solución Afectada	CONFIGURA			
	Responsable	Aprobado por	Consultados	Informados
	DESARROLLO SOFTWARE ECONOMICO-FINANCIERO (NTTDATA) ³	LUIS MARTINEZ FONTIVEROS ⁴	DESARROLLO SOFTWARE ECONOMICO-FINANCIERO (NTTDATA) ⁵ CARLOS GONZALEZ SANTIAGO ⁶ LUIS MARTINEZ FONTIVEROS ⁷	DESARROLLO SOFTWARE ECONOMICO-FINANCIERO (NTTDATA) ⁸ MARIA ANGELES MERINO PAREDES ⁹

Asunto a Decidir	Debemos de elegir la mejor estrategia técnica para la creación y ejecución de clases dinámicas en JPA Opciones: - EntityGraph - Creación dinámica de clases con anotaciones (recarga del EntityManager) - Creación dinámica del persiste.xml (recarga del EntityManager) - NativeQuery

3 https://ws001.sspa.juntadeandalucia.es/confluence/display/~SC00_FEF
4 <https://ws001.sspa.juntadeandalucia.es/confluence/display/~martinezluis70q>
5 https://ws001.sspa.juntadeandalucia.es/confluence/display/~SC00_FEF
6 <https://ws001.sspa.juntadeandalucia.es/confluence/display/~GONZALEZCARLOSA22D>
7 <https://ws001.sspa.juntadeandalucia.es/confluence/display/~martinezluis70q>
8 https://ws001.sspa.juntadeandalucia.es/confluence/display/~SC00_FEF
9 <https://ws001.sspa.juntadeandalucia.es/confluence/display/~MERINOMARIAA25V>

Identificación de la decisión	 Describe brevemente la decisión arquitectónica tomada. Describe brevemente la decisión arquitectónica tomada.
Contexto	Configura, dispone de un ms en su arquitectura pegado al backend que queremos replicar y en la base de datos de sólo lectura. Desde arquitectura se busca una solución configurable en infraestructura sin necesidad de reimplementaciones para cada nuevo backend con el que integrar. En esencia el mecanismo buscado debe resolver: • Configurar el microservicio de sincronía contra la base de datos habitualmente relacional sin necesidad de reimplementaciones. • Asegurar la reutilización de los contenedores implementados. • Delegar el desarrollo de cualquier elemento de configura en un equipo independiente del negocio integrado en configura, evitando dependencias entre proyectos y equipos de trabajo, dinamizando la entrega de valor.

Alternativas Consideradas

EntityGraph

Como primer paso para ver la viabilidad del uso de Entity Graph para la creación dinámica de entidades JPA, se ha realizado el estudio de uso buscando información disponible y artículos escritos, así en las versiones de JPA en las que disponemos de esta funcionalidad. A continuación, se redacta un pequeño resumen del estudio realizado:

- Hasta JPA 2.0, para cargar una asociación de entidad, normalmente usábamos FetchType.LAZY y FetchType.EAGER como estrategias de búsqueda. Esto indica al proveedor JPA que a la hora de realizar una búsqueda se traiga los datos relacionados o no. Desafortunadamente, **esta metaconfiguración es estática** y no permite cambiar entre estas dos estrategias en tiempo de ejecución.
- JPA 2.1 ha introducido la función Entity Graph como un método de mejora de rendimiento en las operaciones de consultas. Permite definir una "plantilla" la cual nos permite agrupar los campos de base de datos relacionados que queremos recuperar en nuestra consulta y nos permite elegir el tipo de gráfico en tiempo de ejecución.

Aun viendo que el uso Entity Graph no se adapta a lo que se está buscando se ha decidido realizar la siguiente **prueba** con el fin de crear dinámicamente entidad JPA:

Prueba EntityGraph

En este caso se va a mostrar la actualización de un registro de la tabla maestra "Unidades de Medidas".

- **Configuración unidad de persistencia (persistence.xml):** En el siguiente desplegable se muestra el código utilizado.

Unidad de persistencia (persistence.xml)

```
<persistence-unit name="jpa-unit-poc" transaction-type="JTA">
  <jta-data-source>jdbc/eventjpadasource</jta-data-source>
  <class>es.ja.csalud.sas.configura.configuraactualizacionconfiguraciones.infra
structure.database.entity.MaestraEntity</class>
  <shared-cache-mode>NONE</shared-cache-mode>
  <properties>
    <property name="eclipselink.ddl-generation" value="none"/>
    <property name="eclipselink.ddl-generation.output-mode" value="sql-
script" />
    <property name="eclipselink.logging.logger" value="DefaultLogger" />
    <property name="eclipselink.logging.level.sql" value="FINE" />
    <property name="eclipselink.logging.parameters" value="true" />
  </properties>
</persistence-unit>
```

- **Tipo petición:** POST
- **URL:** <http://localhost:9080/configura-actualizacion-configuraciones/maestras>¹⁰
- **JSON:** En el siguiente desplegable se muestra el código utilizado.

Body petición REST

```
{
  "fields": {
    "entry": [
      {
        "key": "version",
        "value": "1"
      },
      {
        "key": "activo",
        "value": "false"
      },
      {
        "key": "codigo",
```

¹⁰ <http://localhost:9080/configura-actualizacion-configuraciones/maestras/nativa>

```

        "value": "SOB"
    },
    {
        "key": "descripcion",
        "value": "Descripcion_Test"
    },
    {
        "key": "fraccionable",
        "value": "false"
    },
    {
        "key": "categoria",
        "value": "Peso"
    },
    {
        "key": "observaciones",
        "value": "Observaciones_Test"
    }
    ]
}

```

- **Entidad JPA de Maestra:** En el siguiente desplegable se muestra el código utilizado.

MaestraEntity

```

@Entity
@Table
public class MaestraEntity extends JpaEntity<Integer> implements Serializable {

    private static final long serialVersionUID = -4252716638287114362L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Integer id;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

}

```

- **Código método update:** Se ha sobrescrito el método update para probar el comportamiento de crear un EntityGraph y sus atributos en tiempo de ejecución. En el siguiente desplegable se muestra el código utilizado.

Actualizar MaestraEntity

```
public void updateEntityGraph(Maestra domain) {
    // Se inicializa un nuevo EntityGraph
    EntityGraph<MaestraEntity> entityGraph = this.jpaContext
        .getEntityManager()
        .createEntityGraph(MaestraEntity.class);

    // Se especificar los atributos de la MaestraEntity que nos han llegado en la
    petición Rest
    Map<String, String> fields = domain.getFields();
    if (null != fields && null != fields.entrySet()) {
        fields.entrySet().isEmpty() {
            for (Map.Entry<String, String> entry : fields.entrySet()) {
                if (null != entry.getKey()) {
                    entityGraph.addAttributeNodes(entry.getKey());
                }
            }
        }

        // Modificamos el nombre de la tabla de la maestra
        ClassDescriptor classDescriptor = this.getClassDescriptor(MaestraEntity.class);

        classDescriptor.setTableName("UNIDAD_MEDIDA");

        // Se realiza la búsqueda por id para su posterior merge
        MaestraEntity entity = this.jpaContext
            .getEntityManager()
            .find(MaestraEntity.class, domain.getId());
        this.jpaContext.getMapper().mapTo(domain, entity);
        this.jpaContext.getEntityManager().merge(entity);
    }
}
```

- **Logs consola:** Se observa en los logs como se han generado los nuevos atributos en el EntityGraph pero no pueden ser identificados con los atributos de la entidad JPA, por lo que se produce un error.

Error consola

```
{
  "date": "2024-02-13T06:54:03.268075900",
  "httpStatus": 500,
```

```

    "issue": [
      {
        "diagnostic": "java.lang.IllegalArgumentException: El atributo
[descripcion] no está presente en el tipo gestionado [class
es.ja.csalud.sas.configura.configuraactualizacionconfiguraciones.infrastructure.datab
ase.entity.MaestraEntity].",
        "severity": "fatal"
      }
    ],
    "text": "Se encontró un error inesperado en el servidor."
  }
}

```

Problemas encontrados:

- El uso de Entity Graph se suscribe al ámbito de consultas de base de datos con de tablas relacionadas, y su propósito es la mejora de su rendimiento.
- No se ha encontrado ninguna referencia ni artículo el cuál mencione el uso de Entity Graph para la creación dinámicas de entidades JPA.
- Además, es necesario que existan los atributos de clase en la entidad JPA.

Si recordamos el propósito del diseño de nuestro microservicio actualizador para la aplicación de Configura son:

- Desarrollar un microservicio que sea fácilmente adaptable a sus datos con el fin de sincronizar los cambios realizados en Configura con cada uno de los sistemas finales.
- Desarrollar un microservicio actualizador para el caso de la modificación de valores de tablas maestras.
- En el microservicio actualizador se deberá dar soporte a las acciones de: agregar valores, modificar valores y eliminar valores.
- El microservicio actualizador deberá ser agnóstico de tecnologías externas tal y como se ha planteado en la actualización de parámetros.

Por lo tanto se puede concluir que:

- Consideramos que no es viable esta alternativa.

Creación dinámica de clases con anotaciones (recarga del EntityManager)

En JPA, las clases definidas en el archivo persistence.xml se cargan durante el inicio de la aplicación, cuando se instancia el contexto de persistencia. Estas clases son las entidades que representan tablas en la base de datos y se utilizan para mapear objetos Java a registros de base de datos y viceversa. El proceso de carga de estas clases se produce cuando se crea el EntityManagerFactory, que generalmente ocurre durante la inicialización de la aplicación o la primera vez que se accede a la capa de persistencia, en nuestro caso.

El EntityManagerFactory en JPA carga las clases Java compiladas, es decir, archivos .class y disponibles en el classpath de la aplicación de las entidades que están definidas en el persistence.xml. Cuando creas un EntityManagerFactory, este escanea las clases de entidades especificadas en la unidad de persistencia y las carga en memoria para ser utilizadas durante la ejecución de la aplicación.

Dichas entidad deben, además, tener correctamente anotada con las anotaciones de JPA, como `@Entity` , `@Id` , y otras según sea necesario.

Click para ver imagen

```

<persistence-unit name="jpa-unit" transaction-type="JTA">
  <jta-data-source>jdbc/eventjpadatasource</jta-data-source>
  <class>es.ja.csalud.sas.configura.configuraactualizacionconfiguraciones.infrastructure.database.entity.MaestraEntity</class>
  <shared-cache-mode>NONE</shared-cache-mode>
  <properties>
    <property name="eclipselink.ddl-generation" value="none"/>
    <property name="eclipselink.ddl-generation.output-mode" value="sql-script" />
    <property name="eclipselink.logging.logger" value="DefaultLogger" />
    <property name="eclipselink.logging.level.sql" value="FINE" />
    <property name="eclipselink.logging.parameters" value="true" />
  </properties>
</persistence-unit>

```

Aun vistos los problemas anteriormente citados y por que se observa que la configuración de entidades JPA es estática, a continuación, se exponen las **pruebas realizadas** con el fin de crear dinámicamente **de los atributos y anotaciones y posteriormente recargar el EntityManager**:

Pruebas realizadas con JPA y EclipseLink

Por continuar con la prueba de concepto realizada en [Actualización de maestras en microservicio actualizador](#)¹¹, se decido probar la recarga del EntityManager. En este caso se va a mostrar la actualización de un registro de la tabla maestra "Unidades de Medidas".

- **Configuración unidad de persistencia (persistence.xml):** En el siguiente desplegable se muestra el código utilizado.

Unidad de persistencia (persistence.xml)

```

<persistence-unit name="jpa-unit-poc" transaction-type="JTA">
  <jta-data-source>jdbc/eventjpadatasource</jta-data-source>
  <class>es.ja.csalud.sas.configura.configuraactualizacionconfiguraciones.infra
structure.database.entity.MaestraEntity</class>
  <shared-cache-mode>NONE</shared-cache-mode>
  <properties>
    <property name="eclipselink.ddl-generation" value="none"/>
    <property name="eclipselink.ddl-generation.output-mode" value="sql-
script" />
    <property name="eclipselink.logging.logger" value="DefaultLogger" />
    <property name="eclipselink.logging.level.sql" value="FINE" />
    <property name="eclipselink.logging.parameters" value="true" />
  </properties>
</persistence-unit>

```

- **Tipo petición:** POST
- **URL:** <http://localhost:9080/configura-actualizacion-configuraciones/maestras>¹²
- **JSON:** En el siguiente desplegable se muestra el código utilizado.

¹¹<https://ws001.sspa.juntadeandalucia.es/confluence/pages/viewpage.action?pageId=172463405#Actualizacióndemaestrasenmicroservicioactualizador-PruebasrealizadasconJPAYEclipseLink>

¹² <http://localhost:9080/configura-actualizacion-configuraciones/maestras/nativa>

Body petición REST

```
{
  "fields": {
    "entry": [
      {
        "key": "version",
        "value": "1"
      },
      {
        "key": "activo",
        "value": "false"
      },
      {
        "key": "codigo",
        "value": "SOB"
      },
      {
        "key": "descripcion",
        "value": "Descripcion_Test"
      },
      {
        "key": "fraccionable",
        "value": "false"
      },
      {
        "key": "categoria",
        "value": "Peso"
      },
      {
        "key": "observaciones",
        "value": "Observaciones_Test"
      }
    ]
  }
}
```

- **Entidad JPA de Maestra:** En el siguiente desplegable se muestra el código utilizado.

MaestraEntity

```
@Entity
@Table
public class MaestraEntity extends JpaEntity<Integer> implements Serializable {
```

```

private static final long serialVersionUID = -4252716638287114362L;

@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE)
private Integer id;

public Integer getId() {
    return id;
}

public void setId(Integer id) {
    this.id = id;
}
}

```

- **Código método update:** Se ha sobrescrito el método update para probar el comportamiento una vez recargado tanto el EntityManagerFactory y EntityManager, reproduciendo el mismo ejemplo de la prueba antes mencionada. En el siguiente desplegable se muestra el código utilizado.

Actualizar MaestraEntity

```

@Override
public void update(Maestra domain) {
    Objects.requireNonNull(domain, "Domain parameter must not be null");
    // Se crea el contexto de persistencia
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa-unit-
poc");
    EntityManager entityManager = emf.createEntityManager();

    // Se obtiene el ClassDescriptor y se personaliza las columnas de bases de
datos
    // obtenidas en la petición y configuración SPI
    ServerSession serverSession = (ServerSession)
JpaHelper.getServerSession(emf);
    ClassDescriptor classDescriptor =
serverSession.getClassDescriptor(MaestraEntity.class);
    this.customizer(classDescriptor, domain.getFields());

    // Se cierra EntityManager, volvemos a escanear la unidad de persistencia
    // y se vuelva a recargar el EntityManager
    entityManager.close();
    emf = Persistence.createEntityManagerFactory("jpa-unit-poc");
    entityManager = emf.createEntityManager();

    // Se realiza la consulta para obtener la entidad para actualizar
posteriormente

```

```
// los registros en base de datos
MaestraEntity entity = entityManager.find(MaestraEntity.class,
domain.getId());
entityManager.merge(entity);
}
```

- **Logs consola:** Se observa en los logs como se han generado los mapping de los nuevos campos de base datos pero al no poder ser identificados con los atributos de la entidad JPA nos da un NullPointerException.

Click para ver imagen

```
<terminated> Liberty Runtime [defaultServer] (feb. 5, 2024 8:12:44 p. m.)
[Info de EL]: connection: 2024-02-05 20:13:17.446--ServerSession(1111134042)--Inicio de sesión de
/file:/C:/Dev/SIGLO/workspace/configura-actualizacion-configuraciones/configura-actualizacion-configuraciones-
boot/target/classes/_jpa-unit-poc correcto
[EL Preciso]: sql: 2024-02-05 20:13:22.244--ServerSession(1111134042)--Connection(542217395)--SELECT ID,
DESCRIPCION, CODIGO, CATEGORIA, OBSERVACIONES, VERSION, ACTIVO, FRACCIONABLE FROM UNIDAD_MEDIDA WHERE (ID = ?)
bind => [1011]
[Aviso de EL]: 2024-02-05 20:13:22.738--UnitOfWork(1755174706)--java.lang.NullPointerException
[Info de EL]: connection: 2024-02-05 20:22:44.402--ServerSession(1111134042)--Cierre de sesión de
/file:/C:/Dev/SIGLO/workspace/configura-actualizacion-configuraciones/configura-actualizacion-configuraciones-
boot/target/classes/_jpa-unit-poc correcto
```

Generando clase java con anotaciones en tiempo de ejecución con reflexión

La generación de una clase java de forma dinámica utilizando reflexión tiempo de ejecución se ha observado inviable, debido a que a la creación de atributos de clase no es posible en tiempo de ejecución. Además como se ha mencionado anteriormente, tampoco se ha observado viable la carga de dicha clase en el EntityManagerFactory, debido a que estas clases mapeadas deben ser compiladas para su utilización.

Generando clase java con anotaciones en tiempo de ejecución con Java Byte Code

Aún viendo la poca viabilidad de generar un clases java en tiempo de ejecución y posterior recarga en el EntityManager, se buscó una opción para la generación de clases con manipulación "Java Byte Code", para ello se buscó información para el sencillo uso con estas librerías:

- **Javassist:** <http://www.javassist.org/>
- **Byte Buddy:** <https://bytebuddy.net/#/>, siendo esta la utilizada en el siguiente prueba.
- **Configuración unidad de persistencia (persistence.xml):** En el siguiente desplegable se muestra el código utilizado.

Unidad de persistencia (persistence.xml)

```
<persistence-unit name="jpa-unit-poc" transaction-type="JTA">
  <jta-data-source>jdbc/eventjpadasource</jta-data-source>
```

```

<class>es.ja.csalud.sas.configura.configuraactualizacionconfiguraciones.infra
structure.database.entity.MaestraEntity</class>
<class>es.ja.csalud.sas.configura.configuraactualizacionconfiguraciones.infra
structure.database.entity.MaestraDynamicEntity</class>
<shared-cache-mode>NONE</shared-cache-mode>
<properties>
  <property name="eclipselink.ddl-generation" value="none"/>
  <property name="eclipselink.ddl-generation.output-mode" value="sql-
script" />
  <property name="eclipselink.logging.logger" value="DefaultLogger" />
  <property name="eclipselink.logging.level.sql" value="FINE" />
  <property name="eclipselink.logging.parameters" value="true" />
</properties>
</persistence-unit>

```

- **Tipo petición:** POST
- **URL:** <http://localhost:9080/configura-actualizacion-configuraciones/maestras>¹³
- **JSON:** En el siguiente desplegable se muestra el código utilizado.

Body petición REST

```

{
  "fields": {
    "entry": [
      {
        "key": "version",
        "value": "1"
      },
      {
        "key": "activo",
        "value": "false"
      },
      {
        "key": "codigo",
        "value": "SOB"
      },
      {
        "key": "descripcion",
        "value": "Descripcion_Test"
      },
      {
        "key": "fraccionable",
        "value": "false"
      }
    ]
  }
}

```

¹³ <http://localhost:9080/configura-actualizacion-configuraciones/maestras/nativa>

```

        "key": "categoria",
        "value": "Peso"
    },
    {
        "key": "observaciones",
        "value": "Observaciones_Test"
    }
]
}
}

```

- **Entidad JPA de Maestra:** En el siguiente desplegable se muestra el código utilizado.

MaestraEntity

```

@Entity
@Table
public class MaestraEntity extends JpaEntity<Integer> implements Serializable {

    private static final long serialVersionUID = -4252716638287114362L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Integer id;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

}

```

- **Código creación clase dinámica:** En siguiente ejemplo del desplegable, se genera una clase "MaestaDynamicEntity", la cuál extiende de "MaestraEntity".

Generación clase dinámica MaestraDynamicEntity

```

private Class<?> crearMaestraDynamicEntity() {
    try {
        // Nombre de la clase a la que se agregarán atributos dinámicamente
        String nombreClase =
"es.ja.csalud.sas.configura.configuraactualizacionconfiguraciones."

```

```

+ "infrastructure.database.entity.MaestraDynamicEntity";

// Crear la clase dinámicamente con Byte Buddy extiende de MaestraEntity
return new ByteBuddy()
    .subclass(MaestraEntity.class)
    .name(nombreClase)
    .defineField("version", String.class)
    .annotateField(AnnotationDescription.Builder.ofType(Column.class)
).build()
    .defineField("activo", String.class)
    .annotateField(AnnotationDescription.Builder.ofType(Column.class)
).build()
    .defineField("codigo", String.class)
    .annotateField(AnnotationDescription.Builder.ofType(Column.class)
).build()
    .defineField("descripcion", String.class)
    .annotateField(AnnotationDescription.Builder.ofType(Column.class)
).build()
    .defineField("descripcionPlural", String.class)
    .annotateField(AnnotationDescription.Builder.ofType(Column.class)
).build()
    .defineField("fraccionable", String.class)
    .annotateField(AnnotationDescription.Builder.ofType(Column.class)
).build()
    .defineField("categoria", String.class)
    .annotateField(AnnotationDescription.Builder.ofType(Column.class)
).build()
    .defineField("observaciones", String.class)
    .annotateField(AnnotationDescription.Builder.ofType(Column.class)
).build()
    .make()
    .load(MaestraJpaRepository.class.getClassLoader(),
ClassLoadingStrategy.Default.WRAPPER)
    .getLoaded();
} catch (Exception e) {
    throw new RuntimeException(e);
}
}

```

- **Código método update:** Se ha sobrescrito el método update para probar el comportamiento a la hora de generar la clase de forma dinámica y su posterior recarga en el EntityManager. En el siguiente desplegable se muestra el código utilizado.

Actualizar MaestraDynamicEntity

```

public void updateDynamic(Maestra domain) {
    try {

```

```

Objects.requireNonNull(domain, "Domain parameter must not be null");

// Creación clase EntityDynamicEntity
Class<?> classMaestraDinamicEntity = this.crearMaestraDynamicEntity();

// Crea una nueva instancia de EntityManagerFactory con las clases de
entidad originales y nuevas
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa-
unit-poc");
EntityManager entityManager = emf.createEntityManager();

// Crear una instancia de la nueva clase
Constructor<?> constructorMaestraDinamicEntity =
classMaestraDinamicEntity.getConstructor();
Object entity = constructorMaestraDinamicEntity.newInstance();

// Obtendremos los datos personalizados de la configuración yaml y ya
desarrollados en otras PoC
ServerSession serverSession = (ServerSession)
JpaHelper.getServerSession(emf);
ClassDescriptor classDescriptor =
serverSession.getClassDescriptor(classMaestraDinamicEntity);
classDescriptor.setTableName("UNIDAD_MEDIDA");

// Recrea el EntityManagerFactory (por ejemplo, después de cambiar
dinámicamente el modelo de datos)
emf = Persistence.createEntityManagerFactory("jpa-unit-poc");

// Cierra el EntityManager anterior
entityManager.close();

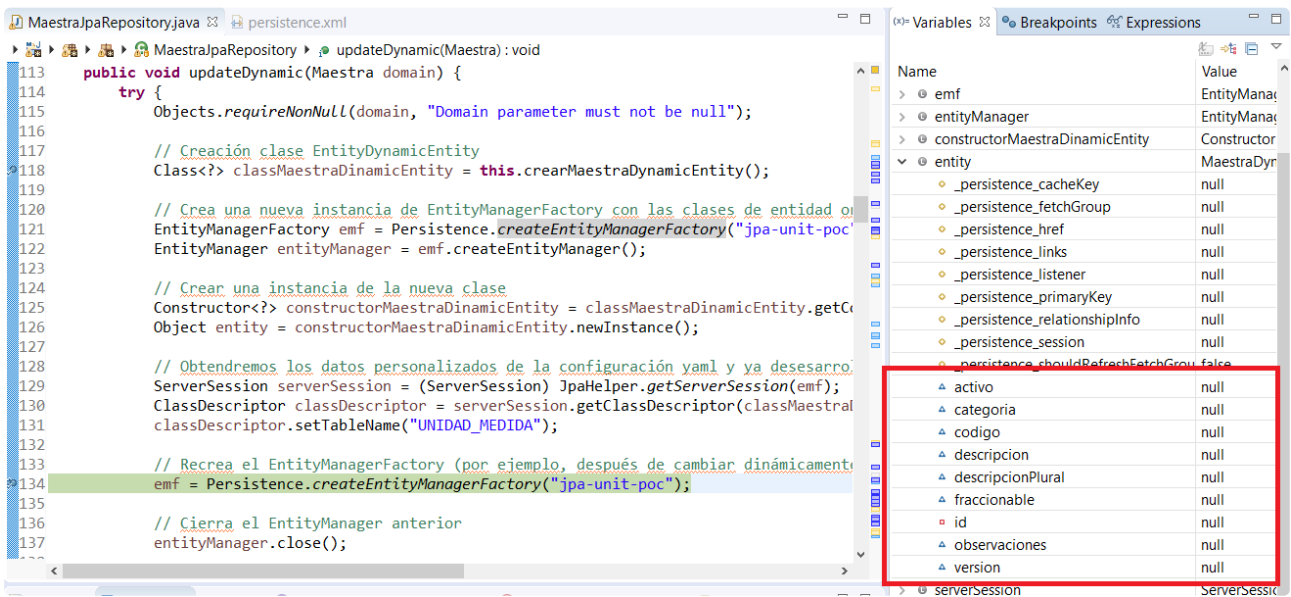
// Obtén un nuevo EntityManager desde la fábrica actualizada
entityManager = emf.createEntityManager();

// Se realiza la búsqueda por id para su posterior merge
entity = entityManager.find(classMaestraDinamicEntity,
domain.getId());
entityManager.merge(entity);
} catch (Exception e) {
    throw new RuntimeException(e);
}
}

```

- **Debug de código:** Para aportar el comportamiento del código expuesto se ha realizado el debugueo del mismo, por lo que se puede llegar a las siguientes conclusiones:
 - La clase "MaestraDynamicEntity" se genera en tiempo de ejecución.

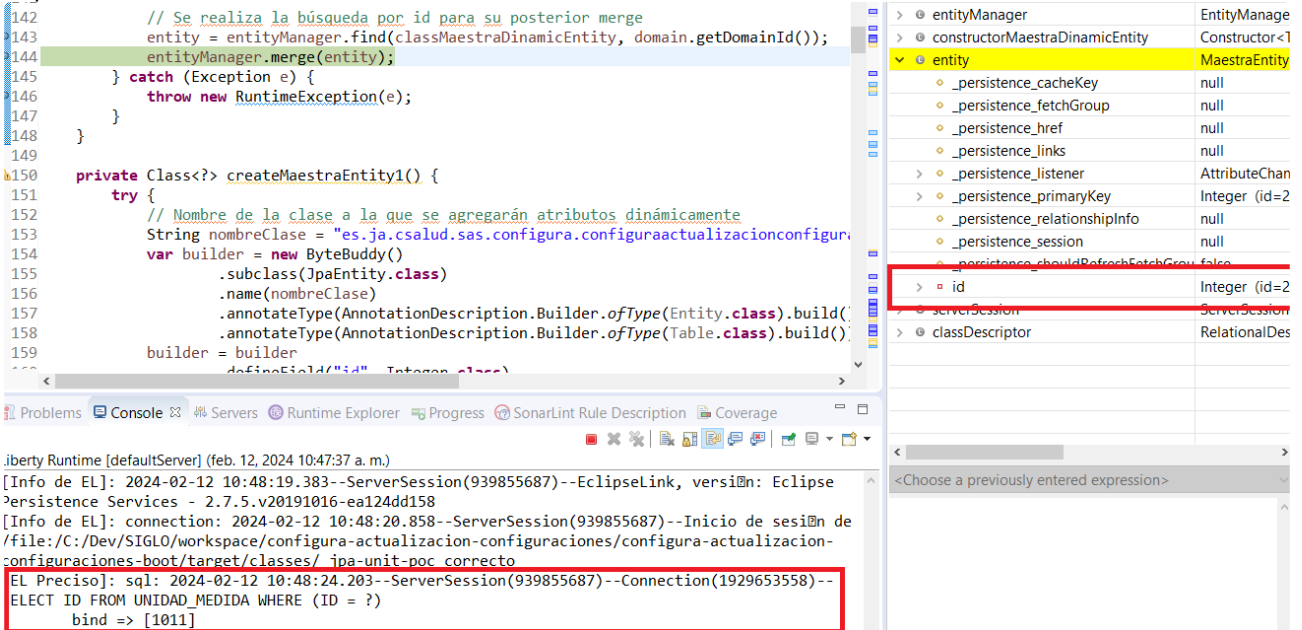
Click para ver imagen



- Tras la recarga del EntityManager y la búsqueda por id del registro para su posterior actualización, se observa que a la hora de realizar la consulta, EntityManagerFactory y EntityManager no son capaces de cargar la nueva clase generada ya que no está disponible en su Classloader.

Click para ver imagen

508px



- Código redefinición clase JPA:** Visto los problemas a hora de generar de forma dinámica en tiempo de ejecución la clase "`MaestraDynamicEntity`", se optó la redefinición de la clase existente "`MaestraEntity`" y su posterior recarga en EntityManager. A continuación, en el desplegable se muestra el código utilizado.

Redefinir clase MaestraEntity

```

private Class<?> redefineMaestraEntity() {
    try {
        return new ByteBuddy()
            .redefine(MaestraEntity.class)
            .defineField("version", String.class)
            .annotateField(AnnotationDescription.Builder.ofType(Column.class)
                .build())
            .defineField("activo", String.class)
            .annotateField(AnnotationDescription.Builder.ofType(Column.class)
                .build())
            .defineField("codigo", String.class)
            .annotateField(AnnotationDescription.Builder.ofType(Column.class)
                .build())
            .defineField("descripcion", String.class)
            .annotateField(AnnotationDescription.Builder.ofType(Column.class)
                .build())
            .defineField("descripcionPlural", String.class)
            .annotateField(AnnotationDescription.Builder.ofType(Column.class)
                .build())
            .defineField("fraccionable", String.class)
            .annotateField(AnnotationDescription.Builder.ofType(Column.class)
                .build())
            .defineField("categoria", String.class)
            .annotateField(AnnotationDescription.Builder.ofType(Column.class)
                .build())
            .defineField("observaciones", String.class)
            .annotateField(AnnotationDescription.Builder.ofType(Column.class)
                .build())
            .make()
            .load(MaestraEntity.class.getClassLoader(),
                ClassLoadingStrategy.Default.WRAPPER)
            .getLoaded();
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

```

- **Log consola:** Al ejecutar dicho código nos da el error que la clase "MaestraEntity" no puede a volver ser cargada, impidiéndonos la posterior recarga en EntityManager.

Error redefinir clase MaestaEntity

```
{
```

```

    "date": "2024-02-12T12:19:53.461987200",
    "httpStatus": 500,
    "issue": [
      {
        "diagnostic": "java.lang.RuntimeException: java.lang.RuntimeException:
java.lang.IllegalStateException: Class already loaded: class
es.ja.csalud.sas.configura.configuraactualizacionconfiguraciones.infrastructure.datab
ase.entity.MaestraEntity",
        "severity": "fatal"
      }
    ],
    "text": "Se encontró un error inesperado en el servidor."
  }
}

```

Problemas encontrados:

- La modificación del mapeo de entidades **JPA utilizando EclipseLink**, es necesario que existan los atributos de clase en la entidad JPA.
- En la utilización de **reflexión** no se ha conseguido añadir atributos y anotaciones en la entidad JPA
- Con la generación de clases entidades JPA con la utilización de **Java Byte Code** vemos los siguientes problemas:
 - Utilización de una librería externa para su uso.
 - No se consigue cargar las clases creadas en el ClassLoader para el uso por la aplicación.

Si recordamos el propósito del diseño de nuestro microservicio actualizador para la aplicación de Configura son:

- Desarrollar un microservicio que sea fácilmente adaptable a sus datos con el fin de sincronizar los cambios realizados en Configura con cada uno de los sistemas finales.
- Desarrollar un microservicio actualizador para el caso de la modificación de valores de tablas maestras.
- En el microservicio actualizador se deberá dar soporte a las acciones de: agregar valores, modificar valores y eliminar valores.
- El microservicio actualizador deberá ser agnóstico de tecnologías externas tal y como se ha planteado en la actualización de parámetros.

Por lo tanto se puede concluir que:

- Consideramos que no es viable esta alternativa.
- Como única opción para su uso sería el análisis del uso de clases por parte de JVM y el ClassLoader y si es posible su modificación tiempo de ejecución.

Creación dinámica del persistence.xml (recarga del EntityManager)

El archivo persistence.xml se utiliza para configurar la unidad de persistencia en una aplicación Java que utiliza JPA. Es un archivo que proporciona información sobre la configuración de la persistencia, como la URL de la base de datos, el nombre del controlador, el nombre del proveedor de persistencia, etc.

El archivo persistence.xml no se puede generar de forma dinámica en tiempo de ejecución, ya que **su contenido es estático y reconocido durante la compilación o el empaquetado de la aplicación**. Además, modificar dinámicamente este archivo puede no ser compatible con el entorno de ejecución.

Sin embargo, la posibilidad de ajustar dinámicamente la configuración de la unidad de persistencia en tiempo de ejecución sería utilizando métodos proporcionados por la JPA Persistence API, con la instanciación de un objeto EntityManagerFactory con propiedades dinámicas, que a continuación se muestra un ejemplo:

Agregar propiedades EntityManagerFactory

```
// Configuración dinámica
Map<String, String> properties = new HashMap<>();
properties.put("javax.persistence.jdbc.url", "jdbc:mysql://localhost:3306/
mydatabase");
properties.put("javax.persistence.jdbc.user", "username");
properties.put("javax.persistence.jdbc.password", "password");

// Crear EntityManagerFactory con propiedades dinámicas
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa-unit",
properties);
EntityManager entityManager = emf.createEntityManager();

// Realizar operaciones a desarrollar
```

Otra posibilidad vista, es la utilización de la etiqueta **<mapping-file>** en el archivo **persistence.xml** se refiere a la etiqueta utilizada para especificar la ubicación de un archivo XML que contiene los metadatos los cuales definen las tablas y campos para las entidades de persistencia.

Cuando usas la etiqueta **<mapping-file>** en el archivo **persistence.xml**, estás indicando a JPA que debe buscar un archivo externo, pero siempre dentro de los ficheros cargados en aplicación, siendo imposible la utilización de dicho mapping con un fichero externo y referenciado a la hora de generar nuestra imagen docker.

El proceso de carga de estas clases se produce cuando durante la inicialización de la aplicación con la implementación de la interfaz **PersistenceProvider**.

Click para ver imagen

```
<persistence-unit name="jpa-unit-orm" transaction-type="JTA">
  <jta-data-source>jdbc/oventipadatasource</jta-data-source>
  <mapping-file>META-INF/unidad-medida-orm.xml</mapping-file>
  <exclude-unlisted-classes>false</exclude-unlisted-classes>
  <shared-cache-mode>NONE</shared-cache-mode>
  <properties>
    <property name="eclipselink.ddl-generation" value="none"/>
    <property name="eclipselink.ddl-generation.output-mode" value="sql-script" />
    <property name="eclipselink.logging.logger" value="DefaultLogger" />
    <property name="eclipselink.logging.level.sql" value="FINE" />
    <property name="eclipselink.logging.parameters" value="true" />
  </properties>
</persistence-unit>
```

Una vez analizado la configuración del **persistence.xml**, a continuación, se exponen las **pruebas realizadas**, aún viendo las dificultades vista a la hora de crear dinámicamente dicho fichero:

Mapeado de tabla maestra utilizando SPI

Como prueba inicial y posible evolución en nuevos pasos a seguir, se ha optado por la configuración por SPI, en la cuál las aplicaciones integradas en Configura, nos faciliten la información de asignación especificada a través de la clase java con los atributos de representativos de las campos de la base de datos y el archivo XML asociados a ellos.

- **Configuración <mapping-file> externo:** Se ha creado un proyecto independiente a configura y aislado de cualquier tecnología tal y como se definió en la esta prueba de concepto [Prueba de concepto: Microservicio actualizador](https://ws001.sspa.juntadeandalucia.es/confluence/display/CONFIGURA/Prueba+de+concepto%3A+Microservicio+actualizador#Pruebadeconcepto:Microservicioactualizador-Aislartecnologiasenmicroservicioactualizador)¹⁴.
- **Fichero persistence-orm.xml:** Se pone como ejemplo el fichero para la maestra de "Unidades de medidas".

persistence-orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings version="2.1"
  xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <package>com.example.entity</package>

  <entity class="UnidadMedidaExternaEntity" name="UNIDAD_MEDIDA">
    <attributes>
      <id name="id" attribute-type="java.lang.Integer">
        <column name="id" />
        <generated-value strategy="SEQUENCE" generator="SQ_PERSONA"/>
        <sequence-generator name = "SQ_PERSONA" sequence-name="SQ_PERSONA" />
      </id>
      <basic name="version" attribute-type="java.lang.String">
        <column name="version" />
      </basic>
      <basic name="activo" attribute-type="java.lang.String">
        <column name="activo" />
      </basic>
      <basic name="codigo" attribute-type="java.lang.String">
        <column name="codigo" />
      </basic>
      <basic name="descripcion" attribute-type="java.lang.String">
        <column name="descripcion" />
      </basic>
      <basic name="descripcionPlural" attribute-type="java.lang.String">
        <column name="descripcionPlural" />
      </basic>
      <basic name="fraccionable" attribute-type="java.lang.String">
```

¹⁴ <https://ws001.sspa.juntadeandalucia.es/confluence/display/CONFIGURA/Prueba+de+concepto%3A+Microservicio+actualizador#Pruebadeconcepto:Microservicioactualizador-Aislartecnologiasenmicroservicioactualizador>

```

        <column name="fraccionable" />
    </basic>
    <basic name="categoria" attribute-type="java.lang.String">
        <column name="categoria" />
    </basic>
    <basic name="observaciones" attribute-type="java.lang.String">
        <column name="observaciones" />
    </basic>
</attributes>
</entity>
</entity-mappings>

```

- **Clase java externa:** A continuación, mostramos el ejemplo de la clase java utilizada para esta prueba, donde implementa la interfaz `ExternalEntity` definida en `configura-api`.

UnidadMedidaExternaEntity.java

```

public class UnidadMedidaExternaEntity implements Serializable, ExternalEntity {

    private static final long serialVersionUID = 4381542316662535995L;

    private Integer id;
    private String version;
    private String activo;
    private String codigo;
    private String descripcion;
    private String descripcionPlural;
    private String fraccionable;
    private String categoria;
    private String observaciones;

    // Getters y setters
}

```

- **Fichero con la información de las clases que implementan ExternalEntity:** Para que las clases desarrolladas puedan ser detectadas por el microservicio actualizador, será necesario incluir un fichero en el proyecto. El fichero debe ubicarse en la ruta **src/main/resources/META-INF/services** y deberá tener el siguiente formato.
 - **Nombre:** `es.ja.csalud.sas.configura.configuraapi.config.spi.ExternalEntity` (Es el nombre de la interfaz que implementan las clases).
 - **Contenido:** Ruta completa de nuestras clases.

Contenido fichero con la información de las clases que implementan ExternalEntity

```
com.example.entity.UnidadMedidaExternaEntity
```

- **Tipo petición:** POST
- **URL:** <http://localhost:9080/configura-actualizacion-configuraciones/maestras>¹⁵
- **JSON:** En el siguiente desplegable se muestra el código utilizado.

Body petición REST

```
{
  "fields": {
    "entry": [
      {
        "key": "version",
        "value": "1"
      },
      {
        "key": "activo",
        "value": "false"
      },
      {
        "key": "codigo",
        "value": "S0B"
      },
      {
        "key": "descripcion",
        "value": "Descripcion_Test"
      },
      {
        "key": "fraccionable",
        "value": "false"
      },
      {
        "key": "categoria",
        "value": "Peso"
      },
      {
        "key": "observaciones",
        "value": "Observaciones_Test"
      }
    ]
  }
}
```

- **Configuración unidad persistencia en persistence.xml de Configura Actualizador:** Donde hacemos referencia al archivo persistence-orm.xml, ya cargado en la aplicación.

¹⁵ <http://localhost:9080/configura-actualizacion-configuraciones/maestras/nativa>

Configura Actualizador persistencia.xml

```

<persistence-unit name="jpa-unit-orm-externa" transaction-type="JTA">
  <jta-data-source>jdbc/eventjpadasource</jta-data-source>
  <mapping-file>META-INF/persistence-orm.xml</mapping-file>
  <exclude-unlisted-classes>false</exclude-unlisted-classes>
  <shared-cache-mode>NONE</shared-cache-mode>
  <properties>
    <property name="eclipselink.ddl-generation" value="none"/>
    <property name="eclipselink.ddl-generation.output-mode" value="sql-
script" />
    <property name="eclipselink.logging.logger" value="DefaultLogger" />
    <property name="eclipselink.logging.level.sql" value="FINE" />
    <property name="eclipselink.logging.parameters" value="true" />
  </properties>
</persistence-unit>

```

- **Configuración server.xml de Configura Actualizador:** La configuración del microservicio actualizador en Microprofile para que cargue las clases externas es bastante sencillo, basta con declarar en el fichero **server.xml** una nueva librería con la ruta en la que posteriormente se incorporarán los jars externos, así como indicar en el classLoader que cargue dichos ficheros en el inicio de la aplicación.

Configura Actualizador server.xml

```

<library id="customConverter" apiTypeVisibility="spec, ibm-api, third-party">
  <fileset dir="/config/external-jars" includes="*.jar" />
</library>

<webApplication
  location="configura-actualizacion-configuraciones.war"
  type="war"
  context-root="configura-actualizacion-configuraciones">
  <classloader privateLibraryRef="customConverter" apiTypeVisibility="spec,
ibm-api, third-party"/>
</webApplication>

```

- **Código método update:** Se ha sobrescrito el método update para probar el comportamiento a la hora de que las aplicaciones que integren en Configura dispongan de la utilización de SPI para el mapeo de entidades JPA. En el siguiente desplegable se muestra el código utilizado.

Actualizar UnidadMedida SPI

```
@Override
```

```

public void updateOrmExterna(Maestra domain) {
    Objects.requireNonNull(domain, "Domain parameter must not be null");
    try {
        // Se obtiene la instancia del clase mapeada por SPI
        Class<?> classUnidadMedidaEntity =
Class.forName("com.example.entity.UnidadMedidaExternaEntity");
        Constructor<?> constructorUnidadMedidaEntity =
classUnidadMedidaEntity.getConstructor();
        ExternalEntity entity = (ExternalEntity)
constructorUnidadMedidaEntity.newInstance();

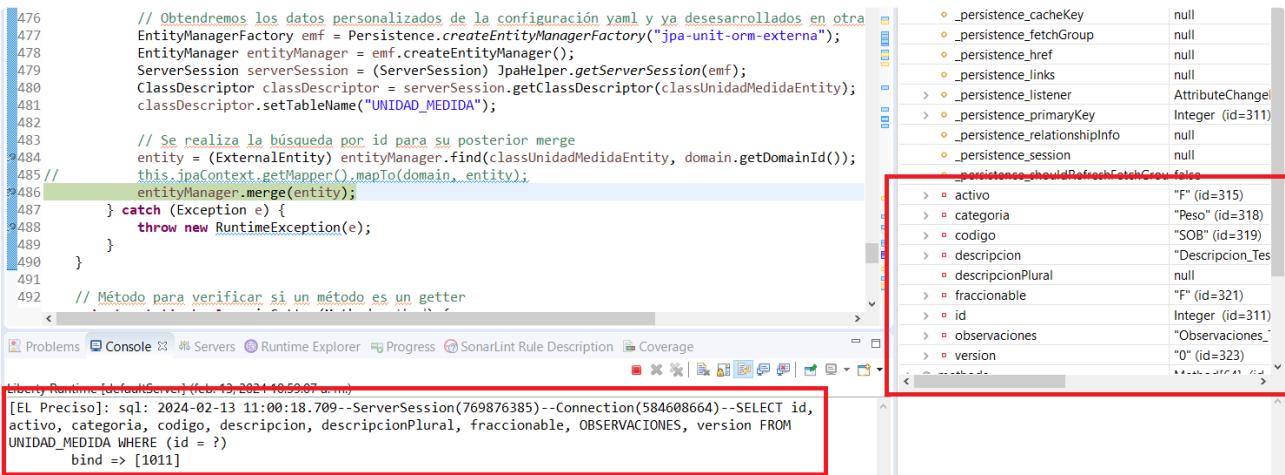
        // Obtendremos los datos personalizados de la configuración yaml y ya
desarrollados en otras PoC
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa-
unit-orm-externa");
        EntityManager entityManager = emf.createEntityManager();
        ServerSession serverSession = (ServerSession)
JpaHelper.getServerSession(emf);
        ClassDescriptor classDescriptor =
serverSession.getClassDescriptor(classUnidadMedidaEntity);
        classDescriptor.setTableName("UNIDAD_MEDIDA");

        // Se realiza la búsqueda por id para su posterior merge
        entity = (ExternalEntity) entityManager.find(classUnidadMedidaEntity,
domain.getId());
        // Necesitamos un mapeador utilizando reflexión para dar valores a los
datos a actualizar
        entityManager.merge(entity);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

```

- **Log consola:** Se observa que funciona correctamente la búsqueda por id en la tabla maestra de "Unidad de Medida".

Click para ver imagen



Problemas encontrados:

- El archivo persistence.xml no generarlo en tiempo de ejecución ya que es un fichero estático
- Se ha observado que se puede facilitar la configuración a través de SPI por las aplicaciones que se integren en Configura, con el inconveniente de la complejidad programática.
- También se observa que necesitaríamos creación de mapeadores complejos en microservicio actualizador de Configura para el mapeo de valores a persistir.

Si recordamos el propósito del diseño de nuestro microservicio actualizador para la aplicación de Configura son:

- Desarrollar un microservicio que sea fácilmente adaptable a sus datos con el fin de sincronizar los cambios realizados en Configura con cada uno de los sistemas finales.
- Desarrollar un microservicio actualizador para el caso de la modificación de valores de tablas maestras.
- En el microservicio actualizador se deberá dar soporte a las acciones de: agregar valores, modificar valores y eliminar valores.
- El microservicio actualizador deberá ser agnóstico de tecnologías externas tal y como se ha planteado en la actualización de parámetros.

Por lo tanto se puede concluir que:

- Consideramos que no es viable esta alternativa.
- Se podría investigar la generación del persistence-orm.xml en tiempo de ejecución al arrancar la aplicación por lo cual sólo necesitaríamos la definición de la clase java representativa de la tabla maestra por SPI, pero por las pruebas realizadas se ve difícil su uso.

NativeQuery

Según se pudo demostrar la PoC realizada [Actualización de maestras en microservicio actualizador](#)¹⁶, se optó por buscar una alternativa en el caso que se necesite sacar un producto viable lo antes posible. En dichas pruebas se ha comprobado que las funcionalidades ya desarrolladas eran compatibles con esta alternativa:

- Modificación de un valor de parámetros
- Validaciones y gestiones de errores
- Aislamiento de cualquier tipo de tecnología

¹⁶ <https://ws001.sspa.juntadeandalucia.es/confluence/pages/viewpage.action?pageId=172463405>

- Configuración externa a través de SPI de convertidores personalizados, configuración tipos de datos y nombre campos, así como la configura de nombre de tablas.

A continuación, ponemos unas de las **pruebas realizadas** en prueba de concepto anteriormente mencionada para que quede registro de la misma:

Prueba generación SQL Nativas en tiempo de ejecución

Es esta prueba se muestra la funcionalidad de crear un registro en la tabla maestra de "Unidades de Medidas".

- **Preparación de entornos:** Se ha preparado un entorno para pruebas realizadas que consiste en los siguientes apartados:
 - **Base de datos:** Como ejemplo de ha copiado la estructura de la tabla de unidades de medida de SIGLO.

UNIDAD_MEDIDA: Created: 11/12/2023 16:42:42 Last DDL: 17/01/2024 9:07:18

Primary Key: <None>

Column Name	I.	PK	Index Pos	Null?	Data Type	Default	Def. On Null	Histogram	Num Distinct	Num Nulls	Density	Collation	Encryption Alg	Salt	Seq/Trigger	Virtual
ID	1		1, 1	N	NUMBER (10)		☐	None	0	0	0			☐	☐	☐
VERSION	2			Y	NUMBER (10)		☐	None	0	0	0			☐	☐	☐
ACTIVO	3			N	CHAR (1 Byte)	T	☐	None	0	0	0	USING_NLS_C OMP		☐	☐	☐
CODIGO	4		1, 2	N	VARCHAR2 (3 Byte)		☐	None	0	0	0	USING_NLS_C OMP		☐	☐	☐
DESCRIPCION	5			N	VARCHAR2 (50 Byte)		☐	None	0	0	0	USING_NLS_C OMP		☐	☐	☐
DESCRIPCIONPLURAL	6			Y	VARCHAR2 (50 Byte)		☐	None	0	0	0	USING_NLS_C OMP		☐	☐	☐
FRACCIONABLE	7			N	CHAR (1 Byte)	F	☐	None	0	0	0	USING_NLS_C OMP		☐	☐	☐
CATEGORIA	8			Y	VARCHAR2 (15 Byte)		☐	None	0	0	0	USING_NLS_C OMP		☐	☐	☐
OBSERVACIONES	9			Y	VARCHAR2 (50 Byte)		☐	None	0	0	0	USING_NLS_C OMP		☐	☐	☐

- **Convertidor externo:** Se ha utilizado un convertidor externo para el tipo de datos Booleano.

```
6
7 public class BooleanToStringConverter implements ConfiguraConverter<String> {
8
9     private static final String INPUT_TRUE = "true";
10    private static final String INPUT_FALSE = "false";
11    private static final String OUTPUT_TRUE = "T";
12    private static final String OUTPUT_FALSE = "F";
13
14    @Override
15    public String convert(Object value) {
16        if (Objects.isNull(value)
17            || (!INPUT_TRUE.equalsIgnoreCase(value.toString()))
18            && !INPUT_FALSE.equalsIgnoreCase(value.toString())) {
19            throw new IllegalArgumentException("The value to be converted is not valid");
20        }
21        return INPUT_TRUE.equalsIgnoreCase(value.toString()) ? OUTPUT_TRUE : OUTPUT_FALSE;
22    }
23
24 }
```

- Configuración fichero yaml

Fichero configuración yaml

```
tableName: UNIDAD_MEDIDA
id:
  strategy: SEQUENCE
  sequenceName: SQ_UNIDAD_MEDIDA
attributeToFieldMapping:
  active:
    fieldName: activo
    fieldClassConverter: com.example.converter.BooleanToStringConverter
  fraccionable:
    fieldClassConverter: com.example.converter.BooleanToStringConverter
```

- **Crear un registro de una maestra:** En esta prueba se ha creado un nuevo registro en la tabla maestra de "Unidades de Medidas".
 - **Tipo petición:** POST
 - **URL:** <http://localhost:9080/configura-actualizacion-configuraciones/maestras/nativa>
 - **JSON:** En el siguiente desplegable se muestra el código utilizado.

Body petición REST

```
{
  "fields": {
    "entry": [
      {
        "key": "version",
        "value": "0"
      },
      {
        "key": "activo",
        "value": "false"
      },
      {
        "key": "codigo",
        "value": "SOB"
      },
      {
        "key": "descripcion",
        "value": "Descripcion_Test"
      },
      {
        "key": "fraccionable",
        "value": "false"
      }
    ]
  }
}
```

- **Logs consola:** Se observa en los logs como se genera valor para el identificado único y posteriormente ejecuta la sentencia SQL de insertar los valores recibidos en nuestro JSON.

- **Resultado base de datos:** Se observa que se ha generado un nuevo registro en la tabla de UNIDAD_MEDIDA.

Problemas encontrados:

- Si recordamos el propósito del diseño de nuestro microservicio actualizador para la aplicación de Configura son:

- Por lo tanto se puede concluir que:**

- ADR - Configura - 000001 - 30

Desde NTT Data, se propone optar por la alternativa de las NativeQuery, puesto que las demás opciones no se han visto viables tras las pruebas de concepto realizadas.

Consecuencias

Desde NTT Data, se plantea como consecuencia de las alternativas vistas que la opción de NativeQuery es la vía a seguir ya que se ajusta a los objetivos descritos inicialmente:

- El microservicio se adapta a modelos de datos relacionales sin necesidad de reimplementaciones. Las aplicaciones que se integren en CONFIGURA únicamente tendrán que aportar sus ficheros de configuración.
- Se dispone de todos los casos de uso requeridos: agregar, modificar y eliminar valores de una tabla maestra.
- Solo se habrá despliegues con las nuevas configuraciones dadas por las aplicaciones integradas en CONFIGURA.
- Nuevas necesidades que apliquen de forma genérica a CONFIGURA, se abordarían por parte del equipo de CONFIGURA. La responsabilidad de las aplicaciones que se integren en CONFIGURA será únicamente indicar su configuración concreta.

Estado Actual

Describe brevemente la decisión arquitectónica tomada.

Actualiza el estado actual de la decisión y si ha habido algún cambio o evolución.