Partner ORACLE UNIVERSITY

Certifica TIC Beyond Certification

INICIO

Desarrollo de Aplicaciones Empresariales con Spring Framework Core 5

ISC. Ivan Venor García Baños

ATC Advanced Training Consultants

SCRUMstudy Authorized Training Partner

Agenda

1. Presentación
2. Objetivos
3. Contenido
4. Despedida

3. Contenido

i. Introducción a Spring Framework
ii. Spring Core
iii. Spring AOP
iv. Spring JDBC – Transaction
v. Spring ORM – Hibernate 5
vi. Spring Data JPA
vii. Fundamentos Spring MVC y Spring REST
viii. Fundamentos Spring Security
ix. Seguridad en Servicios REST
x. Introducción Spring Boot

vi. Spring Data JPA

vi. Spring Data JPA (a)

vi.i Introducción

a. ¿Qué es Spring Data?

b. Módulos Spring Data

vi.ii Spring Data JPA

a. Dependencias

b. Configuración

c. Repositorios Spring Data

Práctica h. Parte 1 - Configuración capa Repository Spring Data JPA

vi. Spring Data JPA (b)

vi.iii Implementación capa de datos con Spring Data JPA

a. Operaciones CRUD

b. Derived Queries

c. Paginación y Ordenamiento

d. @Query Methods

e. Implementaciones personalizadas

Práctica h. Parte 2 – Implementación capa Repository Spring Data JPA

vi. Spring Data JPA (c)

vi.iv Otros repositorios Spring Data

a. Spring Data MongoDB

Práctica i. Repositorios Spring Data MongoDB

b. Spring Data JDBC

Práctica j. Repositorios Spring Data JDBC

vi.i Introducción

Objetivos de la lección

vi.i Introducción

- Revisar a grandes rasgos qué es Spring Data.
- Revisar los diferentes módulos que componen el proyecto de Spring Data.
- Comprender los beneficios de implementar Spring Data en proyectos Java Empresariales.

vi.i Introducción

a. ¿Qué es Spring Data?

b. Módulos Spring Data

vi.i Introducción (a)

- ¿Qué es Spring Data?

- Spring Data es un proyecto base que utiliza el modelo de programación de Spring Framework para unificar y facilitar el acceso a datos.

- Spring Data es una colección de proyectos de acceso a datos los cuales utilizan una misma abstracción (común) para ejecutar operaciones sobre los distintos repositorios, sean SQL como NoSQL.

- Spring Data ofrece múltiples implementaciones de acceso a datos para los diversos motores de persistencia.

vi.i Introducción (b)

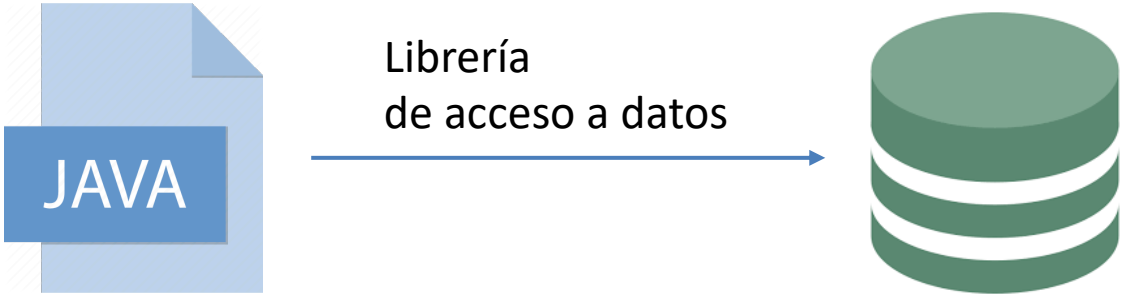
- ¿Qué es Spring Data?

- En lo general, toda aplicación Java empresarial requerirá implementar una capa de acceso a datos para persistir la información que maneja.

- Comúnmente, los programadores al desarrollar múltiples aplicaciones acceso a datos, consiguen la capacidad de implementar la capa de datos utilizando diferentes tecnologías como JPA, MongoDB, JDBC, Redis, etcétera, lo cuál ocasiona que éstos deban conocer con precisión el ciclo de vida de dichos frameworks de persistencia.

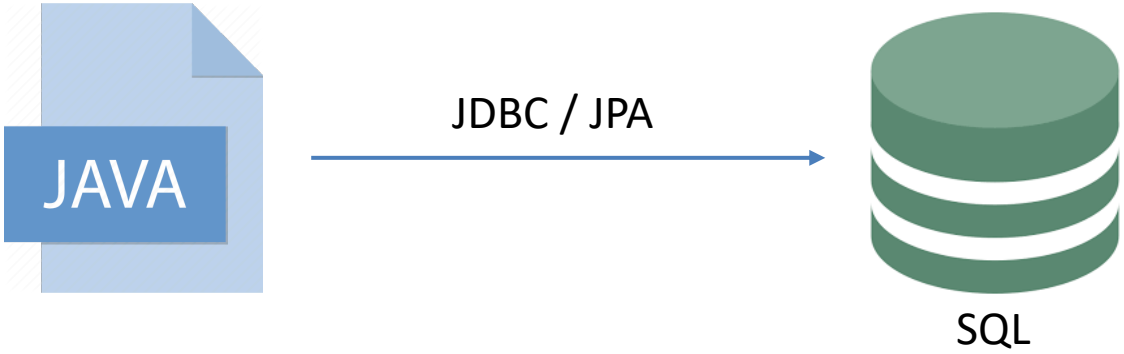
vi.i Introducción (c)

- Aplicaciones Java sin Spring Data.



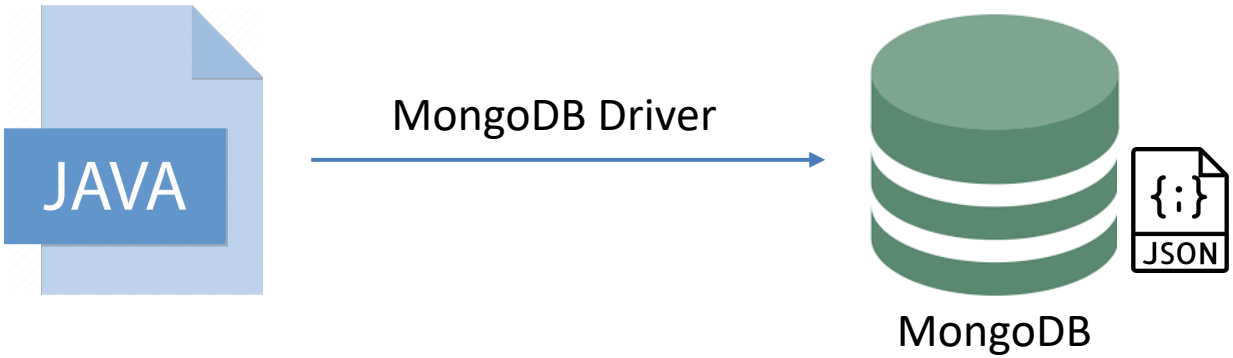
vi.i Introducción (d)

- Aplicaciones Java sin Spring Data.



vi.i Introducción (e)

- Aplicaciones Java sin Spring Data.



vi.i Introducción (f)

- ¿Qué es Spring Data?

- La codificación de la capa de datos, es una de las tareas de desarrollo más repetitivas en la implementación de aplicaciones Java empresariales lo cuál, también, ocasiona que sean las tareas con mayor propensión a introducir errores.

- Spring Data ofrece clases e interfaces que facilitarán el desarrollo de la capa de acceso a datos, de aplicaciones basadas en Spring, donde los desarrolladores no tendrán que implementar de nueva cuenta DAOs o repositorios.

vi.i Introducción (g)

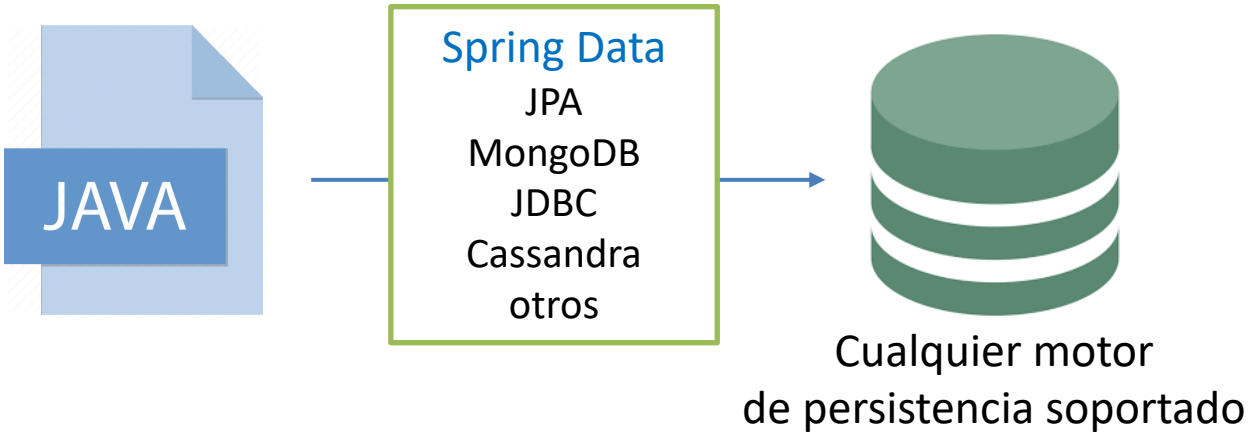
- ¿Qué es Spring Data?

- La misión de Spring Data, es proveer un mecanismo familiar y consistente para que los desarrolladores puedan implementar repositorios de forma rápida y fácil a través de múltiples motores de persistencia como JDBC, JPA, LDAP, MongoDB, Redis, Cassandra, Geode, Solr, Gemfire, CouchBase, Elasticsearch, Neo4J, Hadoop, entre otros.

- Aprendiendo el modelo de programación de Spring Data, se ofrecerá la posibilidad de implementar acceso a datos sobre cualquiera de los motores de acceso a datos sin necesidad de conocerlos.

vi.i Introducción (h)

- Aplicaciones Java con Spring Data.



vi.i Introducción (i)

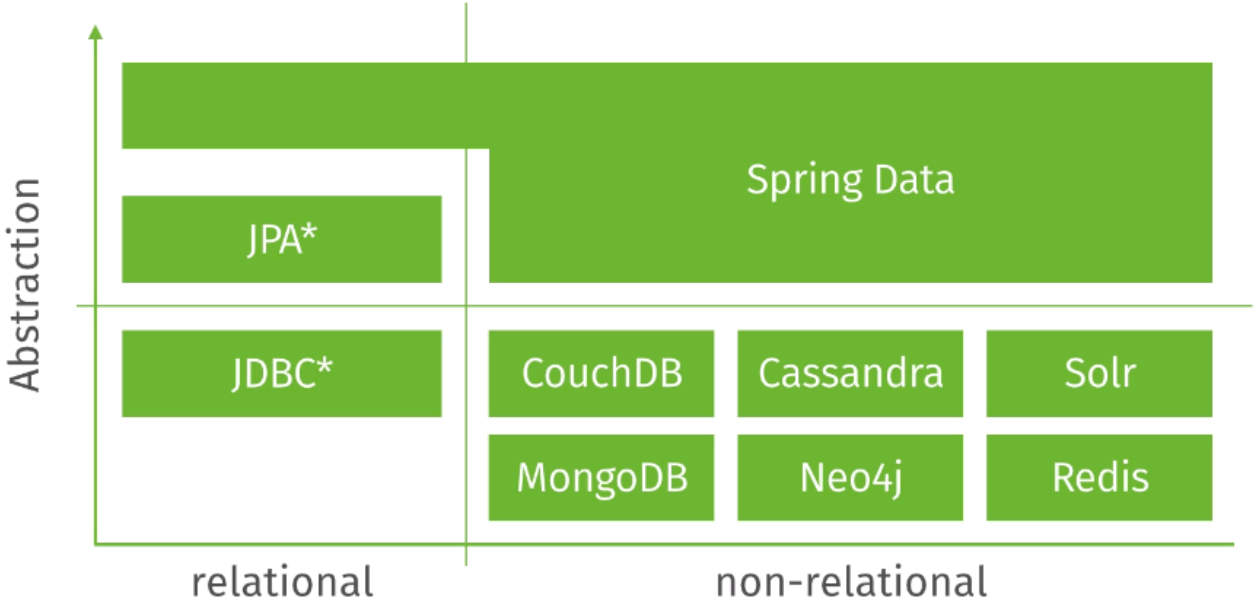
- ¿Qué es Spring Data?

- Spring Data reducirá:
  - Código al implementar DAOs o repositorios (por tanto elimina la necesidad de conocer los frameworks de persistencia y elimina la propensión a errores).
  - Reduce código de plomería (*boilerplate code*).

- Spring Data facilitará:
  - La implementación de consultas (*queries*) de forma declarativa (a través del nombrado de los métodos que definamos, siguiendo un conjunto de reglas).

vi.i Introducción (j)

- ¿Qué es Spring Data?



Confidencial 2020

vi.i Introducción

a. ¿Qué es Spring Data?

b. Módulos Spring Data

vi.i Módulos Spring Data (a)

- Spring Data, es un proyecto *umbrella* el cual cobija proyectos especializados en un motor de base de datos (relacional o no relacional) particular.

- Existen proyectos Spring Data oficiales y no oficiales los cuales son soportados por la comunidad de usuarios.

vi.i Módulos Spring Data (b)

- Proyectos oficiales Spring Data (a).

- **Spring Data Commons**: Clases e interfaces base que dan soporte a todos los demás módulos de Spring Data.

- **Spring Data JDBC**: Soporte para la implementación de repositorios basados en JDBC.

- **Spring Data JPA**: Soporte para implementaciones basadas en el estándar JPA (requiere la implementación JPA correspondiente como Hibernate).

vi.i Módulos Spring Data (c)

- Proyectos oficiales Spring Data (b).

- **Spring Data KeyValue**: Soporte para la implementación de repositorios de tipo llave-valor en memoria basados en mapas (Maps).

- **Spring Data LDAP**: Soporte para la integración hacia repositorios LDAP Linux o Microsoft Active Directory mediante Lightweight Directory Access Protocol.

- **Spring Data MongoDB**: Soporte para la definición de repositorios basados en MongoDB.

vi.i Módulos Spring Data (d)

- Proyectos oficiales Spring Data (c).

- **Spring Data Redis**: Soporte para la implementación de repositorios de tipo llave-valor en memoria basados en Redis.

- **Spring Data REST**: Proyecto que exporta los repositorios como recursos RESTful mediante HATEOAS (Hypermedia as the Engine of Application State).

- Entre otros como Spring Data para **Apache Cassandra, Apache Geode, Apache Solar y Pivotal Gemfire**.

vi.i Módulos Spring Data (e)

- Proyectos no oficiales Spring Data (community) (a).

- Spring Data Aerospike
- Spring Data ArangoDB
- Spring Data Couchbase
- Spring Data Azure Cosmos (Microsoft Azure Cosmos DB).
- Spring Data Cloud Datastore (Google Datastore).
- Spring Data Cloud Spanner (Google Spanner).
- Spring Data DynamoDB (Amazon DynamoDB).
- Spring Data Elasticsearch .
- Spring Data Hazelcast (*In memory Data Grid*).

vi.i Módulos Spring Data (f)

- Proyectos no oficiales Spring Data (community) (b).

- Spring Data Jest (Elasticsearch con soporte para el cliente Jest REST).
- Spring Data Neo4j (Soporte para grafos mediante Neo4j).
- Spring Data Vault (Soporte para repositorios de tipo “baúl”, construido sobre el módulo Spring Data KeyValue).

vi.i Módulos Spring Data (g)

- Spring Data *Umbrella*.

Confidencial 2020

Resumen de la lección

vi.i Introducción

- Revisamos a grandes rasgos qué es el proyecto Spring Data y por qué debemos utilizarlo.
- Revisamos los diferentes módulos que componen el proyecto de Spring Data, tanto oficiales como no oficiales.
- Comprendimos la implicación del desarrollo de la capa de acceso a datos en aplicaciones Java empresariales.
- Comprendimos los beneficios de implementar Spring Data en proyectos basados en Spring Framework.

Esta página fue intencionalmente dejada en blanco.

vi.ii Spring Data JPA

Objetivos de la lección

vi.ii Spring Data JPA

- Comprender que es Spring Data JPA.
- Aprender el patrón de diseño Repository o *Repository Pattern*.
- Analizar el beneficio de utilizar repositorios Spring Data.
- Aprender a configurar Spring Data JPA.
- Implementar repositorios de entidades JPA mediante las interfaces *repository* que implementa Spring Data JPA.

vi.ii Spring Data JPA

a. Dependencias
b. Configuración
c. Repositorios Spring Data
Práctica h. Parte 1 - Configuración capa Repository Spring Data JPA

vi.ii Spring Data JPA (a)

- Spring Data JPA es uno de los proyectos más populares del ecosistema Spring Data debido a que habilita la posibilidad de crear repositorios JPA (basados en entidades JPA) de una manera rápida y fácil.

- El módulo Spring Data JPA implementa un avanzado soporte para el manejo de entidades JPA a través de sus mapeos los cuales pueden estar definidos en configuración XML o por medio de @Anotaciones javax.persistence tales como **@Entity**, **@Table**, **@Column**, **@OneToMany** o **@ManyToOne**.

vi.ii Spring Data JPA (b)

- Al igual que todos los módulos Spring Data, Spring Data JPA facilita el desarrollo de la capa de acceso a datos, hacia los motores de bases de datos relacionales soportados por medio de JPA.

- Mediante Spring Data JPA se reduce significativamente el esfuerzo que se requiere para trabajar con el objeto EntityManager de la especificación JPA.

- Spring Data JPA permite la inyección del objeto EntityManager a través de la anotación **@PersistenceContext** sin mayor configuración.

vi.ii Spring Data JPA (c)

- Spring Data JPA ofrece (a):
  - Soporte para implementar repositorios de capa de acceso a datos a bases de datos relacionales mediante el estándar JPA a través de interfaces comunes.
  - Soporte de predicados QueryDSL para implementar consultar JPA type-safe.
  - Auditoría para la detección de creación y modificación entidades de forma transparente al desarrollador.
  - Paginación.
  - Ejecución dinámica de consultar (queries) a través de consultas derivadas o "*derived queries*".

vi.ii Spring Data JPA (d)

- Spring Data JPA ofrece (b):
  - Soporte para la inyección de EntityManager a través de la anotación **@PersistenceContext** e implementar consultas personalizadas.
  - Soporte para la implementación de consultas personalizadas a través de anotaciones mediante **@Query** mediante JPQL.
  - Soporte para la ejecución de consultas nativas (SQL).
  - Mapeo de entidades por medio de anotaciones o XML.
  - Configuración de repositorios mediante **@EnableJpaRepositories**.
  - Soporte transaccional mediante Spring-tx y anotación **@Transactional**.

vi.ii Spring Data JPA

a. Dependencias

b. Configuración

c. Repositorios Spring Data

Práctica h. Parte 1 - Configuración capa Repository Spring Data JPA

vi.ii Dependencias (a)

- Para configurar un proyecto Spring Data JPA es requerido agregar las dependencias:
  - Spring Data JPA, la cual agrega Spring Data Commons.
  - Implementación JPA, Hibernate por ejemplo.
  - Driver de base de datos; H2 o MySQL, etc.
  - Implementación DataSource con Pool de Conexiones; DBCP, C3P0, Hikari, etc.

- Para configurar el soporte transaccional, agregar las dependencias:
  - Spring-Tx, Spring AOP
  - AspectJ (aspectjrt y aspectjweaver).

vi.ii Dependencias (b)

- Coordenadas de dependencias Maven (a).

- Spring Data JPA:
  org.springframework.data:spring-data-jpa:${spring-data-jpa.version}

- Hibernate:
  org.hibernate:hibernate-entitymanager:${hibernate-version}
  org.hibernate:hibernate-validator:${hibernate-validator-version}

- H2:
  com.h2database:h2:${com.h2database-version}

vi.ii Dependencias (c)

- Coordenadas de dependencias Maven (b).

- DataSource C3P0:
  c3p0:c3p0:${c3p0.version}

- Spring TX:
  org.springframework:spring-tx

- Spring AOP:
  org.springframework:spring-aop

vi.ii Dependencias (d)

- Coordenadas de dependencias Maven (c).

- AspectJ:
  org.aspectj:aspectjrt:${aspectj-version}
  org.aspectj:aspectjweaver:${aspectj-version}

vi.ii Spring Data JPA

a. Dependencias

b. Configuración

c. Repositorios Spring Data

Práctica h. Parte 1 - Configuración capa Repository Spring Data JPA

vi.ii Configuración (a)

- Para habilitar Spring Data JPA en un proyecto Java empresarial es necesario realizar las siguientes configuraciones.

- Sobre una clase de configuración **@Configuration** Java Config o mediante configuración por XML, definir DataSource.

```
@Bean
public DataSource dataSource() {
    return new EmbeddedDatabaseBuilder()
                .generateUniqueName(true)
                .setType(EmbeddedDatabaseType.H2)
                .build();
}
```

```
<jdbc:embedded-database
            id="datasource" type="H2">
```

vi.ii Configuración (b)

- Definición de DataSource productivo (ejemplo):

```
@Bean
public DataSource dataSource() {
    BasicDataSource dataSource = new BasicDataSource();
    dataSource.setDriverClassName(driverClassName);
    …
    return dataSource();
}
```

```
<bean id="datasource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="${db.driverClassName}" />
    <property name="url" value="${db.url}" />
    <property name="username" value="${db.user}" />
    <property name="password" value="${db.pass}" />
</bean>
```


Confidencial 2020

vi.ii Configuración (c)

- Definición de **EntityManagerFactory** a través de la clase **LocalContainerEntityManagerFactoryBean**, implementación de FactoryBean<EntityManagerFactory> (a).

```
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource datasource) {
    LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();
    em.setDataSource(datasource);
    em.setPackagesToScan(new String[] { "package.to.scan.mapped.entities" });

    JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    em.setJpaVendorAdapter(vendorAdapter);
    em.setJpaProperties(additionalProperties());
    return em;
}
```

Inyección de DataSource

Escaneo de clases @Entity

Otras configuraciones de Hibernate (proveedor JPA)

Confidencial 2020

vi.ii Configuración (d)

- Definición de **EntityManagerFactory** a través de la clase **LocalContainerEntityManagerFactoryBean**, implementación de FactoryBean<EntityManagerFactory> (b).

```
Properties additionalProperties() {
    Properties properties = new Properties();
    properties.setProperty("hibernate.hbm2ddl.auto", "create-drop");
    properties.setProperty("hibernate.show_sql", "true");
    …
    properties.setProperty("hibernate.dialect", "org.hibernate.dialect.H2Dialect");
    return properties;
}
```

Modo de configuración DDL

Mostrar SQL en consola (útil para debug)

Dialecto JDBC

Confidencial 2020

vi.ii Configuración (e)

- Es posible, configurar el **EntityManagerFactory** a través de la clase **LocalEntityManagerFactoryBean** en lugar de **LocalContainerEntityManagerFactoryBean** sin embargo, únicamente cambia el modo en el que Spring construye el EntityManagerFactory en base a la especificación JPA.

- **LocalEntityManagerFactoryBean**, configura un EntityManagerFactory basado en la especificación de aplicación.

- **LocalContainerEntityManagerFactoryBean**, configura un EntityManagerFactory basado en la especificación de contenedor.

vi.ii Configuración (f)

- Realizar el mapeo de entidades en el paquete especificado durante la configuración del **LocalContainerEntityManagerFactoryBean**.

```
package package.to.scan.mapped.entities;

@Entity
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String name;
    private Integer credits;
    …
}
```

vi.ii Configuración (g)

- Habilita la configuración de repositorios Spring Data JPA mediante la anotación **@EnableJpaRepositories**.

- La anotación **@EnableJpaRepositories** requiere el nombre del paquete base que contendrá los repositorios Spring Data JPA:
  @EnableJpaRepositories(
  basePackages="package.to.scan.repositories")

- De forma análoga, es posible habilitar la configuración de repositorios Spring Data JPA mediante XML a través de la etiqueta:
  <**jpa:repositories** base-package="package.to.scan.repositories"/>.

vi.ii Configuración (h)

- Opcional, es posible agregar la configuración de Spring Tx, para habilitar transaccionabilidad SQL, a través de la anotación **@EnableTransactionManagement**.

- En caso de habilitar el soporte transaccional de Spring Tx, es necesario registrar un bean de tipo **JpaTransactionManager** el cual es implementación de la interface **PlatformTransactionManager**.

- Por último, definir los repositorios Spring Data JPA en el paquete definido sobre la anotación **@EnableJpaRepositories**. La definición de los repositorios Spring Data JPA se revisará en la sección siguiente.

vi.ii Spring Data JPA

a. Dependencias
b. Configuración
c. Repositorios Spring Data
Práctica h. Parte 1 - Configuración capa Repository Spring Data JPA

vi.ii Repositorios Spring Data (a)

- Spring Data promueve la implementación del patrón de diseño Repositorio o "*Repository Pattern*".

- El patrón de diseño *Repository* son clases, interfaces y/o componentes que abstraen y encapsulan la lógica requerida para acceder a un repositorio de datos particular.

- El *Repository Pattern* define una interfaz o abstracción agnóstica del motor de persistencia lo cual significa que, entre los métodos que define, no incluye información relevante del motor de persistencia al que accede.

vi.ii Repositorios Spring Data (b)

- El patrón de diseño *Repository* sirve de fachada hacia la aplicación para ocultar el detalle técnico requerido para implementar la interacción y/o comunicación hacia el repositorio de datos concreto.

- En otras palabras, el patrón de diseño *Repository* es "*una abstracción de acceso a datos agnóstica del motor de persistencia*".

- Por lo general, el patrón de diseño *Repository* define interfaces con métodos comunes de acceso a datos tal como operaciones CRUD y, las implementaciones de estas interfaces implementan las operaciones definidas hacia un motor de persistencia concreto.

vi.ii Repositorios Spring Data (c)

- Spring Data implementa el patrón de diseño *Repository* a través de un conjunto de interfaces las cuales definen el comportamiento genérico de acceso a datos que, los proveedores de los tipos de repositorios soportados deberán implementar.

- Las interfaces principales, de Spring Data Commons, que definen el patrón *Repository* son:
  - **Repository**
  - **CrudRepository**
  - **PagingAndSortingRepository**

vi.ii Repositorios Spring Data (d)

- Spring Data *Repository Pattern* (a).

- Interfaz **Repository**:
  Interface marcadora o *marker interface* que identifica el tipo del objeto de dominio a persistir (clase de la entidad) y el tipo del objeto identificador o Id de la entidad (los identificadores no pueden ser primitivos en Spring Data).

```
// Interface marcadora, para "marcar" Repositorios Spring Data
public interface Repository<T, ID> {

}
```

vi.ii Repositorios Spring Data (e)

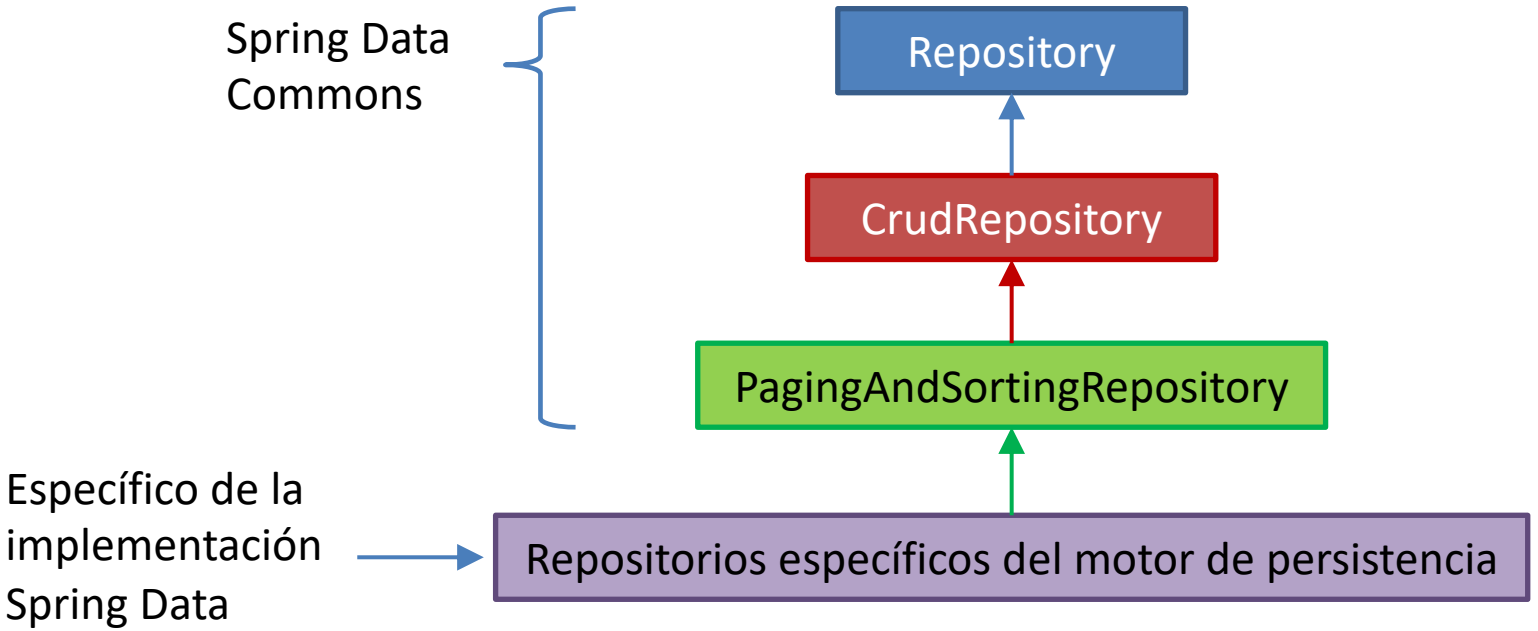
- Spring Data *Repository Pattern* (b).

- Interfaz **CrudRepository**:
  Interface genérica que define operaciones CRUD para un repositorio específico. La interface **CrudRepository** es agnóstica del motor de persistencia.

- Interfaz **PagingAndSortingRepository**:
  Interface extensión de **CrudRepository** que provee métodos adicionales con funcionalidades de paginación y ordenamiento.

vi.ii Repositorios Spring Data (f)

- Spring Data *Repository Pattern* (c).



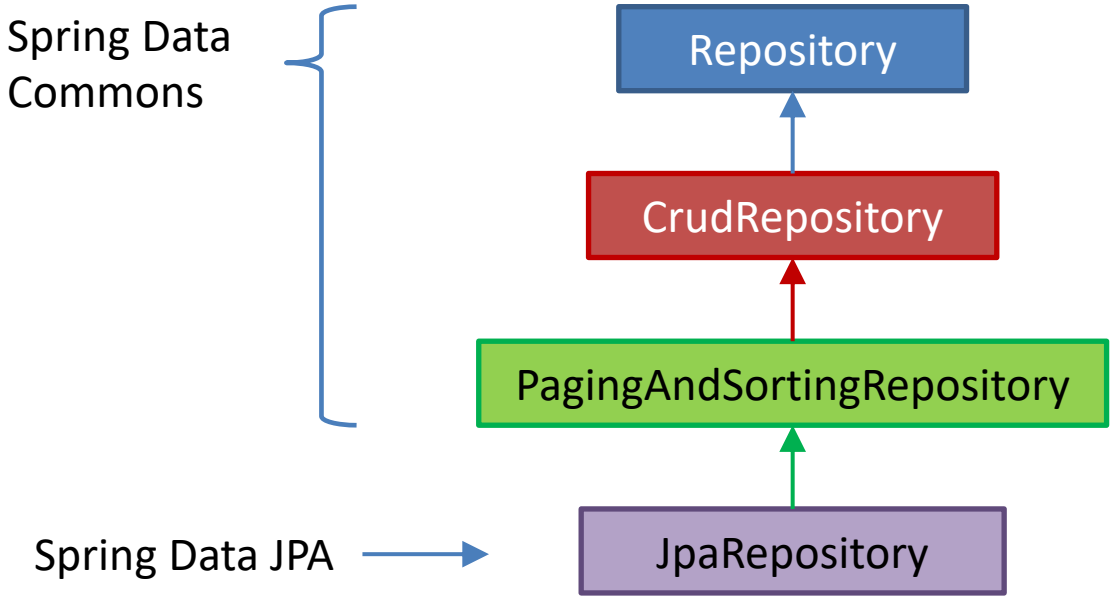
vi.ii Repositorios Spring Data (g)

- Spring Data JPA es el módulo Spring Data que implementa las abstracciones **Repository**, **CrudRepository** y **PagingAndSortingRepository** para acceso a datos a repositorios SQL a través del estándar JPA.

- Por otro lado, Spring Data JPA, provee la interface **JpaRepository**, la cual extiende de **PagingAndSortingRepository** y define funcionalidades propias de JPA como framework de persistencia.
  - Operaciones como **flush()**, **saveAndFlush()** o **deleteInBatch()** son operaciones implementadas por el estándar JPA.

vi.ii Repositorios Spring Data (h)

- Spring Data *Repository Pattern* implementación JPA.



Confidencial

vi.ii Repositorios Spring Data (i)

- Definición de Beans Spring Data JPA *Repositories*.

- Para definir un bean *repository* con Spring Data JPA es necesario:
  - Definir los beans requeridos para configurar Spring Data.
  - Definir una las interfaces *repository* que extienda de **CrudRepository**, **PagingAndSortingRepository** o **JpaRepository**.
    - Definir los tipos genéricos T y ID, en donde T es la clase de la Entidad a persistir y ID es la clase de la llave primaria de la Entidad a persistir.
  - Las interfaces *repository* definidas deben contenerse en el paquete indicado sobre la anotación **@EnableJpaRepositories**.

vi.ii Repositorios Spring Data (j)

- Definición de Bean CourseRepository.

```
package package.to.scan.repositories;

public interface CourseRepository
    extends CrudRepository<Course, Long> {
}

public interface CourseRepository
    extends PagingAndSortingRepository< Course, Long> {
}

public interface CourseRepository
    extends JpaRepository< Course, Long> {
}
```

```
package package.to.scan.mapped.entities;

@Entity
public class Course {
    @Id
    @GeneratedValue
    private Long id;

    private String name;
    private Integer credits;
    …
}
```

vi.ii Spring Data JPA

a. Dependencias
b. Configuración
c. Repositorios Spring Data
Práctica h. Parte 1 - Configuración capa Repository Spring Data JPA

vi.ii Spring Data JPA. Práctica h. (a)

- Práctica h – Parte 1. Configuración capa Repository Spring Data JPA.

- Comprender cómo definir los beans necesarios para configurar Spring Data JPA.
- Aprender a definir los beans correspondientes a la implementación de JPA con Hibernate.
- Aprender a configurar transaccionabilidad mediante Spring Tx en Spring Data JPA.
- Revisar el mapeo de entidades JPA y definir repositorios con Spring Data JPA.

Resumen de la lección

vi.ii Spring Data JPA

- Comprendimos lo que es Spring Data Commons y Spring Data JPA.
- Aprendimos el patrón de diseño Repository o *Repository Pattern*.
- Analizamos el beneficio de utilizar repositorios Spring Data JPA y comprendimos como utilizarlo en ambientes transaccionales.
- Aprendimos a configurar Spring Data JPA a través de anotaciones.
- Implementamos repositorios de entidades JPA mediante las interfaces CrudRepository, PagingAndSortingRepository y JpaRepository que implementa Spring Data JPA.

Esta página fue intencionalmente dejada en blanco.

vi.iii Implementación capa de datos con Spring Data JPA

Objetivos de la lección

vi.iii Implementación capa de datos con Spring Data JPA

- Comprender como desarrollar repositorios mediante Spring Data JPA.
- Implementar operaciones CRUD mediante CrudRepository.
- Comprender e implementar *derived queries* a través de las interfaces *repository* disponibles.
- Comprender e implementar paginación y ordenamiento de consultas a tarvés de las interfaces **Page**, **Pageable** y **Sort**.
- Aprender a definir consultas personalizadas.
- Aprender a definir consultas nativas.

vi.iii Implementación capa de datos con Spring Data JPA

a. Operaciones CRUD
b. Derived Queries
c. Paginación y Ordenamiento
d. @Query Methods
e. Implementaciones personalizadas
Práctica h. Parte 2 – Implementación capa Repository Spring Data JPA

vi.iii Operaciones CRUD (a)

- Spring Data Commons define la interfaz **CrudRepository**, la cual define el comportamiento genérico de operaciones CRUD sobre entidades.

- Spring Data JPA crea, al vuelo (*on the fly*), la implementación correspondiente a la interface que definamos, la cual debe extender de **CrudRepository**.

vi.iii Operaciones CRUD (b)

- Interfaz **CrudRepository**.

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S entity);
    <S extends T> Iterable<S> saveAll(Iterable<S> entities);
    Optional<T> findById(ID id);
    boolean existsById(ID id);
    Iterable<T> findAll();
    Iterable<T> findAllById(Iterable<ID> ids);
    long count();
    void deleteById(ID id);
    void delete(T entity);
    void deleteAll(Iterable<? extends T> entities);
    void deleteAll();
}
```


Confidencial 2020

vi.iii Operaciones CRUD (c)

- Definición de **CourseRepository**, extensión de **CrudRepository**.
- *Course* es una clase entidad que se mapea a una tabla en base de datos, se define mediante anotaciones estándar JPA.
- Long es el tipo del identificador o id de la entidad *Course*.

```
public interface CourseRepository extends CrudRepository<Course, Long> {

}
```

vi.iii Operaciones CRUD (d)

- Operaciones CRUD básicas:

```
@Autowired
CourseRepository courseRepository;

// Crear y Actualizar (insert y update)
Course course = courseRepository.save(new Course("English"));

// Recuperar (retrieve)
Optional<Course> retrievedCourse = courseRepository.findById(1L);
Iterable<Course> allCourses = courseRepository.findAll();

// Eliminar (delete)
void courseRepository.deleteById(course.getId());
void courseRepository.delete(course);
```

vi.iii Implementación capa de datos con Spring Data JPA

a. Operaciones CRUD
b. Derived Queries
c. Paginación y Ordenamiento
d. @Query Methods
e. Implementaciones personalizadas
Práctica h. Parte 2 – Implementación capa Repository Spring Data JPA

vi.iii Derived Queries (a)

- Los *queries* derivados o *derived queries* son métodos definidos en una interface *repository* que se traducen a una consulta que se realizará al repositorio de datos correspondiente.

- En otras palabras, los *queries* derivados o *derived queries* es un query o consulta que se genera automáticamente, por Spring Data, mediante la firma del método que la define.

- A través de *queries* derivados es posible definir consultas más complejas, que simples operaciones CRUD, mediante el nombrado de métodos personalizados en una interface *repository*.

vi.iii Derived Queries (b)

- Spring Data JPA implementa un mecanismo dinámico de construcción de *queries* basado en la firma de los métodos definidos en la interface *repository*. El nombrado de los métodos debe seguir un conjunto de reglas y sintaxis.

- Suponga que existe una entidad Persona mapeada con anotaciones de JPA de la siguiente forma:

```
@Entity
public class Person {
    @Id
    @GeneratedValue
    private Long id;
    private String emailAddress;
    private String lastname;
}
```

vi.iii Derived Queries (c)

- Un ejemplo de *derived query* o *query* derivado es el siguiente:

```
public interface PersonRepository extends Repository<Person, Long> {
    List<Person> findByEmailAddressAndLastname(String emailAddress, String lastname);
}
```

- El mecanismo de implementación de *derived queries* de Spring Data JPA analiza la firma de los métodos definidos y verifica si estos comienzan con: **find…By**, **read…By**, **query…By**, **count…By** y **get…By**.

- Si el método comienza con alguno de los prefijos anteriores, Spring Data JPA continúa analizando el resto del nombre del método.

vi.iii Derived Queries (d)

- La información que puede contener el método, que inicia con con **find…By**, **read…By**, **query…By**, **count…By** y/o **get…By**, puede contener la palabra **Distinct**, la cual servirá para habilitar la bandera *dinstinct* en una consulta hacia el repositorio de datos correspondiente.

```
public interface PersonRepository extends Repository<Person, Long> {
    List<Person> findDistinctByEmailAddress(String emailAddress);
}
```

vi.iii Derived Queries (e)

- En su definición, los *derived queries* pueden contener las siguientes palabras reservadas (a):

Palabra	Ejemplo	JPQL
And	**findBy**Lastname**And**Firstname	where x.lastname = ?1 and x.firstname = ?2
Or	**findBy**Lastname**Or**Firstname	where x.lastname = ?1 or x.firstname = ?2
Is,Equals	**findBy**Firstname **findBy**Firstname**Is** **findBy**Firstname**Equals**	where x.firstname = ?1
Between	**findBy**StartDate**Between**	where x.startDate between ?1 and ?2

vi.iii Derived Queries (f)

- En su definición, los derived queries pueden contener las siguientes palabras reservadas (b):

Palabra	Ejemplo	JPQL
LessThan	**findBy**Age**LessThan**	where x.age < ?1
LessThanEqual	**findBy**Age**LessThanEqual**	where x.age <= ?1
GreaterThan	**findBy**Age**GreaterThan**	where x.age > ?1
GreaterThanEqual	**findBy**Age**GreaterThanEqual**	where x.age >= ?1
After	**findBy**StartDate**After**	where x.startDate > ?1
Before	**findBy**StartDate**Before**	where x.startDate < ?1
IsNull	**findBy**Age**IsNull**	where x.age is null
IsNotNull,NotNull	**findBy**Age**(Is)NotNull**	where x.age not null

vi.iii Derived Queries (g)

- En su definición, los derived queries pueden contener las siguientes palabras reservadas (c):

Palabra	Ejemplo	JPQL
Like	**findBy**Firstname**Like**	where x.firstname like ?1
NotLike	**findBy**Firstname**NotLike**	where x.firstname not like ?1
StartingWith	**findBy**Firstname**StartingWith**	where x.firstname like ?1 (1*)
EndingWith	**findBy**Firstname**EndingWith**	where x.firstname like ?1 (2*)
Containing	**findBy**Firstname**Containing**	where x.firstname like ?1 (3*)

1* El parámetro firstname contiene el carácter '%' al final. Ej: "**Iv%**"
2* El parámetro firstname contiene el carácter '%' al inicio. Ej: "**%an**"
3* El parámetro firstname contiene el carácter '%' al inicio y al final. Ej: "**%va%**"

Confidencial 2020

vi.iii Derived Queries (h)

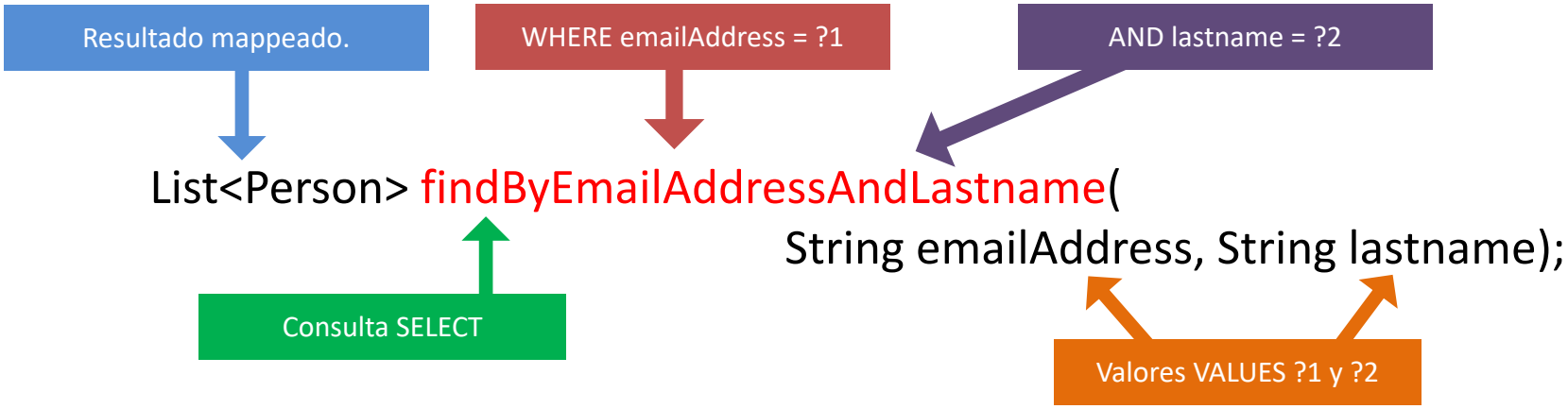
- En su definición, los derived queries pueden contener las siguientes palabras reservadas (d):

Palabra	Ejemplo	JPQL
OrderBy	**findBy**Age**OrderBy**Lastname**Desc**	where x.age = ?1 order by x.lastname desc
Not	**findBy**Lastname**Not**	where x.lastname <> ?1
In	**findBy**Age**In**(Collection<Age> ages)	where x.age in ?1
NotIn	**findBy**Age**NotIn**(Collection<Age> ages)	where x.age not in ?1
True	**findBy**Active**True**()	where x.active = true
False	**findBy**Active**False**()	where x.active = false
IgnoreCase	**findBy**Firstname**IgnoreCas**e	where UPPER(x.firstame) = UPPER(?1)

Confidencial 2020

vi.iii Derived Queries (i)

- *Derived queries*.



Confidencial 2020

vi.iii Derived Queries (j)

- Beneficios (a):
  - No más implementaciones DAO fallidas, no más ”DAOs genéricos”.
  - No más código repetitivo (*boilerplate code* o código de plomería).
  - Implementación de repositorios generados al vuelo.
  - Manejo del API de JPA a bajo nivel por la implementación y no por el desarrollador.
  - Mapeo de resultados automático. No más necesidad de *RowMappers*.

Confidencial 2020

vi.iii Derived Queries (k)

- Beneficios (b):

  - Fácil implementación y línea de aprendizaje
  - No existe necesidad de escribir consultas SQL nativas.
  - Facilidad de intercambiar implementaciones, por ejemplo de Spring Data JPA a Spring Data MongoDB.
  - Spring Data ha sido altamente probado y soportado por la comunidad.

vi.iii Implementación capa de datos con Spring Data JPA

a. Operaciones CRUD
b. Derived Queries
c. Paginación y Ordenamiento
d. @Query Methods
e. Implementaciones personalizadas
Práctica h. Parte 2 – Implementación capa Repository Spring Data JPA

vi.iii Paginación y Ordenamiento (a)

- En la implementación de aplicaciones Java empresariales, los requerimientos de acceso a datos son más complejos que la simple aplicación de operaciones CRUD o *queries* derivados.

- Dependiendo del caso de uso es probable que las aplicaciones lleguen a mantener en sus repositorios de datos miles de millones de registros.
  - ¿Cómo implementarán los desarrolladores la lectura de todos los registros de una tabla para procesar sus datos?
    - ~~Leyéndolos todos y procesarlos en memoria.~~
    - Leyéndolos por partes, lotes o por página.

vi.iii Paginación y Ordenamiento (b)

- La implementación de paginación y ordenamiento es otra funcionalidad que aporta Spring Data de forma automática, para todos los repositorios de datos soportados.

- ¿Cómo definimos una interface genérica, portable que soporte paginación y ordenamiento y, que funcione con cualquier repositorio de datos?

- Spring Data garantiza que la generación automática de paginación y ordenamiento sea agnóstica del repositorio de datos.

vi.iii Paginación y Ordenamiento (c)

- Paginación y ordenamiento SQL Server 2005 y 2008

```
SELECT * FROM (
        SELECT ROW_NUMBER() OVER ( ORDER BY OrderDate ) AS RowNum, *
        FROM Orders
        WHERE OrderDate >= '1980-01-01'
    ) AS RowConstrainedResult
WHERE RowNum >= 1
AND     RowNum < 10
ORDER BY RowNum;
```

vi.iii Paginación y Ordenamiento (d)

- Paginación y ordenamiento SQL Server 2012

```
SELECT * FROM Orders
WHERE OrderDate >= '1980-01-01’
ORDER BY id
OFFSET 1 ROWS FETCH NEXT 10 ROWS ONLY;
```

vi.iii Paginación y Ordenamiento (e)

- Paginación y ordenamiento MySQL 3+

```
SELECT * FROM Orders
WHERE OrderDate >= '1980-01-01’
ORDER BY id
LIMIT 1, 10;
```

vi.iii Paginación y Ordenamiento (f)

- Paginación y ordenamiento MongoDB

```
db.orders
    .find()
    .sort({ orderDate: '1980-01-01' })
    .skip(0)
    .limit(10);
```

vi.iii Paginación y Ordenamiento (g)

- La interface **PagingAndSortingRepository** define métodos comunes con soporte para paginación y ordenamiento.

- Las implementaciones concretas de Spring Data, como Spring Data JPA, generan la implementación concreta de la interface **PagingAndSortingRepository** al vuelo, la cual se basa en las mismas abstracciones que se define en **PagingAndSortingRepository**.

vi.iii Paginación y Ordenamiento (h)

- Interfaz **PagingAndSortingRepository**.

- La interfaz **PagingAndSortingRepository** extiende de **CrudRepository**.

```
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {
    Iterable<T> findAll(Sort sort);
    Page<T> findAll(Pageable pageable);
}
```

vi.iii Paginación y Ordenamiento (i)

- La interfaz **PagingAndSortingRepository** define el uso de las abstracciones **Sort** y **Pageable** para ordenar y paginar resultados.

- Para implementar paginación en un método *repository*, es necesario utilizar la abstracción **Pageable** para que Spring Data devuelva un objeto genérico de tipo **Page** el cual contiene los resultados de la consulta.

- Se utiliza la abstracción **Sort**, para que Spring Data devuelva una página o colección de objetos correspondiente a los resultados de forma ordenada. El ordenamiento se da por medio del motor de persistencia.

vi.iii Paginación y Ordenamiento (j)

- Interfaces **Sort** y **Pageable**.

- **Pageable** es una abstracción que define las reglas que deben de cumplirse para generar la paginación de resultados de la consulta a ejecutar.

- **Sort** es una abstracción que define las reglas que deben cumplirse para ordenar una colección o página de resultados devueltos por una consulta.

vi.iii Paginación y Ordenamiento (k)

- Interface **Page**.

- **Page** es una abstracción que contiene la colección que devuelve la consulta ordenada y/o paginada conforme a la definición del método ejecutado.

- La interface **Page** contiene otra información útil para la paginación como el número total de páginas de la consulta y el número total de elementos que arroja la consulta.

- La Interface **Page** extiende de la interface **Slice**.

vi.iii Paginación y Ordenamiento (I)

- Interface **Page**.

```
public interface Page<T> extends Slice<T> {
    int getTotalPages();
    long getTotalElements();
    <U> Page<U> map(Function<? super T, ? extends U> converter);
}
```

vi.iii Paginación y Ordenamiento (m)

- Interface **Slice**.

```
public interface Slice<T>
                extends Streamable<T> {
    int getNumber();
    int getSize();
    int getNumberOfElements();
    List<T> getContent();
    boolean hasContent();
    Sort getSort();
    boolean isFirst();
    boolean isLast();
    …
    …
    boolean hasNext();
    boolean hasPrevious();
    Pageable nextPageable();
    Pageable previousPageable();
    <U> Slice<U> map(
            Function<? super T, ? extends U> converter);
}
```


Confidencial 2020

vi.iii Paginación y Ordenamiento (n)

- Definición de **CourseRepository**, extensión de **PagingAndSortingRepository**.

```
public interface CourseRepository extends PagingAndSortingRepository<Course, Long> {
    Page<Course> findByName(String name, Pageable pageable, Sort sort);
}
```

vi.iii Paginación y Ordenamiento (ñ)

- Operaciones de paginación y ordenamiento básicas:

```
@Autowired
CourseRepository courseRepository;

// Resultados paginados y ordenados
Page<Course> course = courseRepository.findByName(name,
                      PageRequest.of(pageNumber, pageSize),
                      Sort.by("name"));
```

Valor del atributo "name" utilizado en clausula Where

Envoltura Page de la colección de entidades devueltas por la consulta

Columna de ordenamiento, con dirección de ordenamiento default (ascendente).

Valor del número de página a consultar. La primer página es cero.

Valor del tamaño de la página o elementos por página.

Confidencial 2020

vi.iii Implementación capa de datos con Spring Data JPA

a. Operaciones CRUD
b. Derived Queries
c. Paginación y Ordenamiento
d. @Query Methods
e. Implementaciones personalizadas
Práctica h. Parte 2 – Implementación capa Repository Spring Data JPA

vi.iii @Query Methods (a)

- Los ***@Query** methods* o métodos **@Query** son métodos que se definen en interfaces *repository* y que declaran la consulta o *query* a ejecutar, de forma declarativa por medio de anotaciones.

- La anotación **@Query** se utiliza para definir la consulta a ejecutar por medio de algún lenguaje específico de consultas propietario del Framework ORM como JPQL o por medio de SQL nativo.

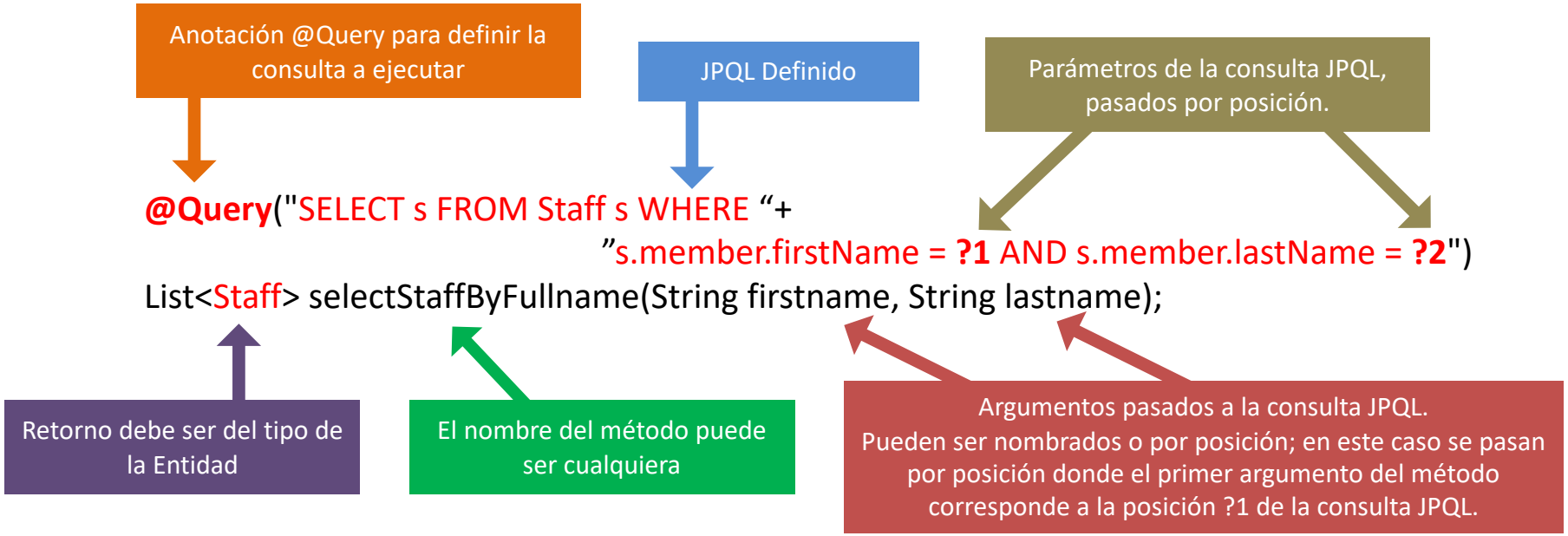
vi.iii @Query Methods (b)

- Cuando se utiliza la anotación **@Query**, para definir la consulta a ejecutar por el repositorio, el nombre del método no debe seguir las reglas pre-establecidas para los *queries* derivados.

- En otras palabras, al utilizar la anotación **@Query**, la firma del método puede ser cualquiera, únicamente se debe respetar el valor de retorno correspondiente a la entidad.

vi.iii @Query Methods (c)

- *@Query methods*.



Confidencial 2020

vi.iii @Query Methods (d)

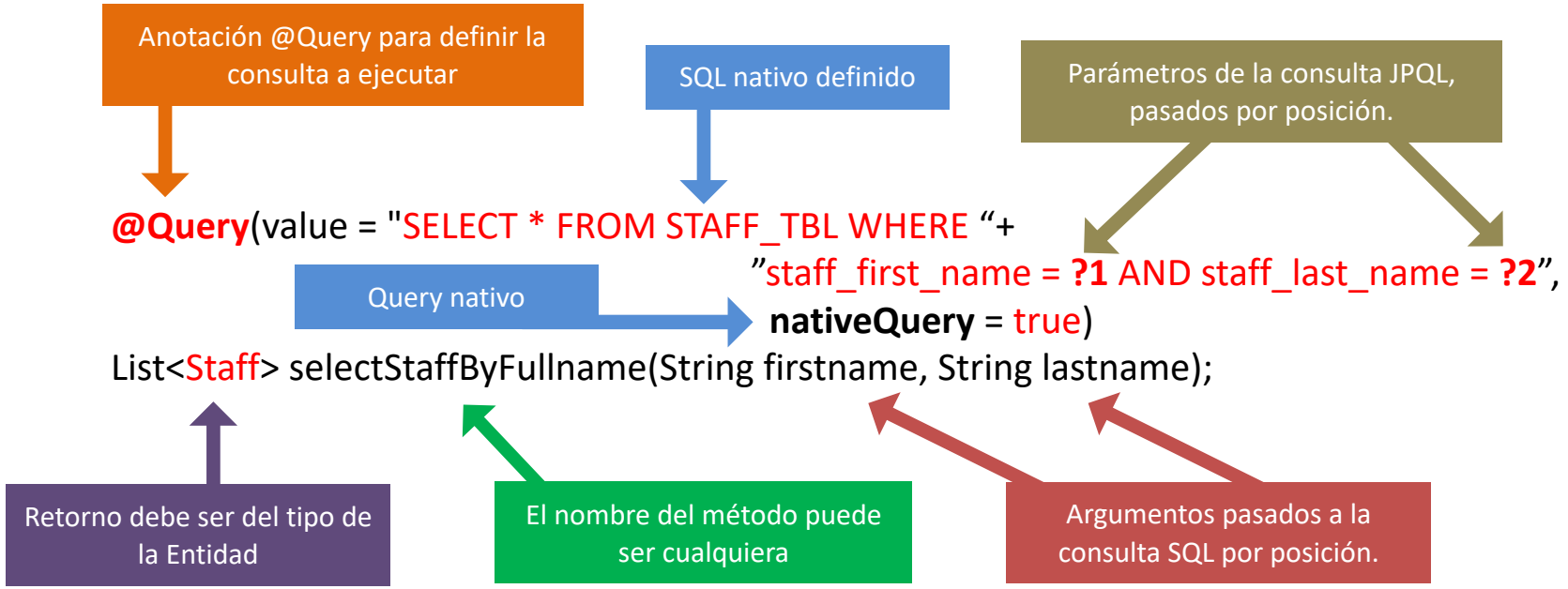
- Habilitando la propiedad **nativeQuery** de la anotación **@Query**, es posible definir *queries* nativos mediante SQL.

- Tanto para consultas JPQL como para *queries* nativos, es posible utilizar parámetros nombrados, en lugar de posicionales.

- Para utilizar parámetros nombrados en consultas JPQL o *queries* nativos es necesario utilizar la anotación **@Param** sobre los argumentos del método definido en la interface *repository*.

vi.iii @Query Methods (e)

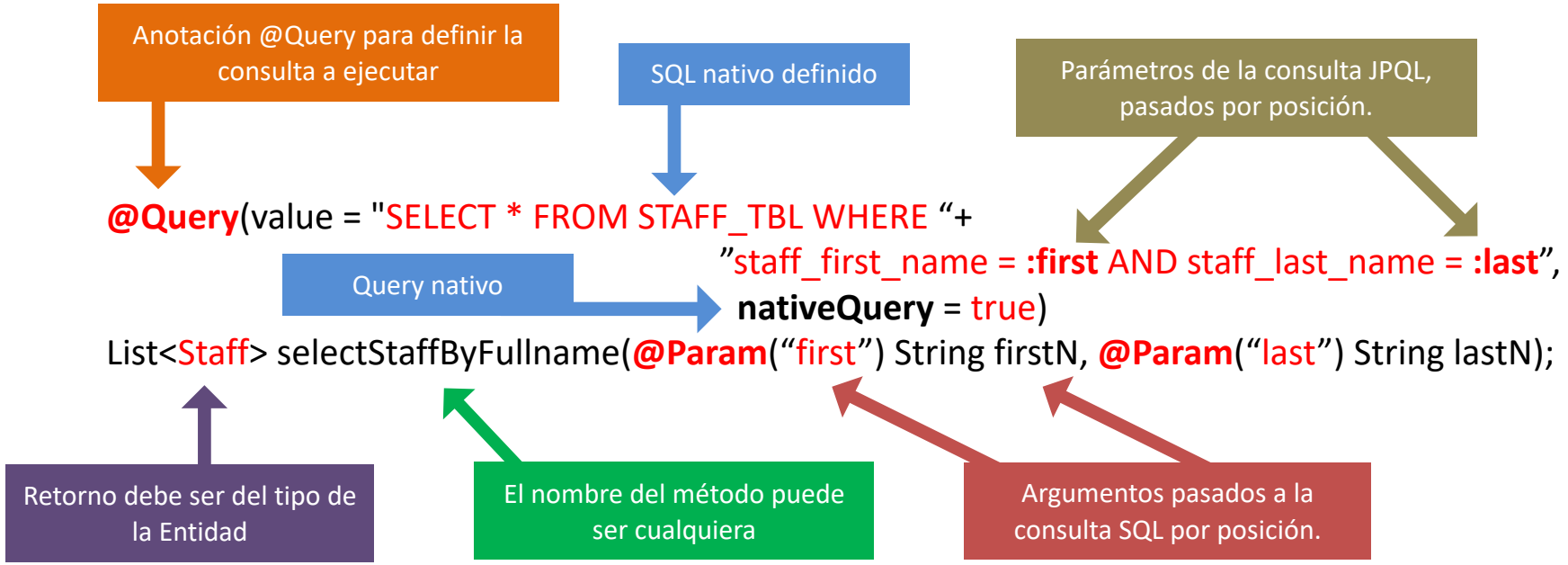
- *Native @Query methods*, parámetros por posición (a).



Confidencial 2020

vi.iii @Query Methods (f)

- *Native @Query methods*, parámetros nombrados (b).



Confidencial 2020

vi.iii Implementación capa de datos con Spring Data JPA

a. Operaciones CRUD
b. Derived Queries
c. Paginación y Ordenamiento
d. @Query Methods
e. Implementaciones personalizadas
Práctica h. Parte 2 – Implementación capa Repository Spring Data JPA

vi.iii Implementaciones personalizadas (a)

- Las implementaciones personalizadas permiten codificar consultas a repositorios de bases de datos utilizando el Framework ORM o librería de persistencia utilizada por Spring Data.

- Es posible codificar implementaciones personalizadas a través de definir una nueva interface donde se defina el método a implementar.

- Para el caso de Spring Data JPA, mediante implementaciones personalizadas podemos trabajar directamente con el objeto **EntityManager** inyectándolo mediante **@PersistenceContext**.

vi.iii Implementaciones personalizadas (b)

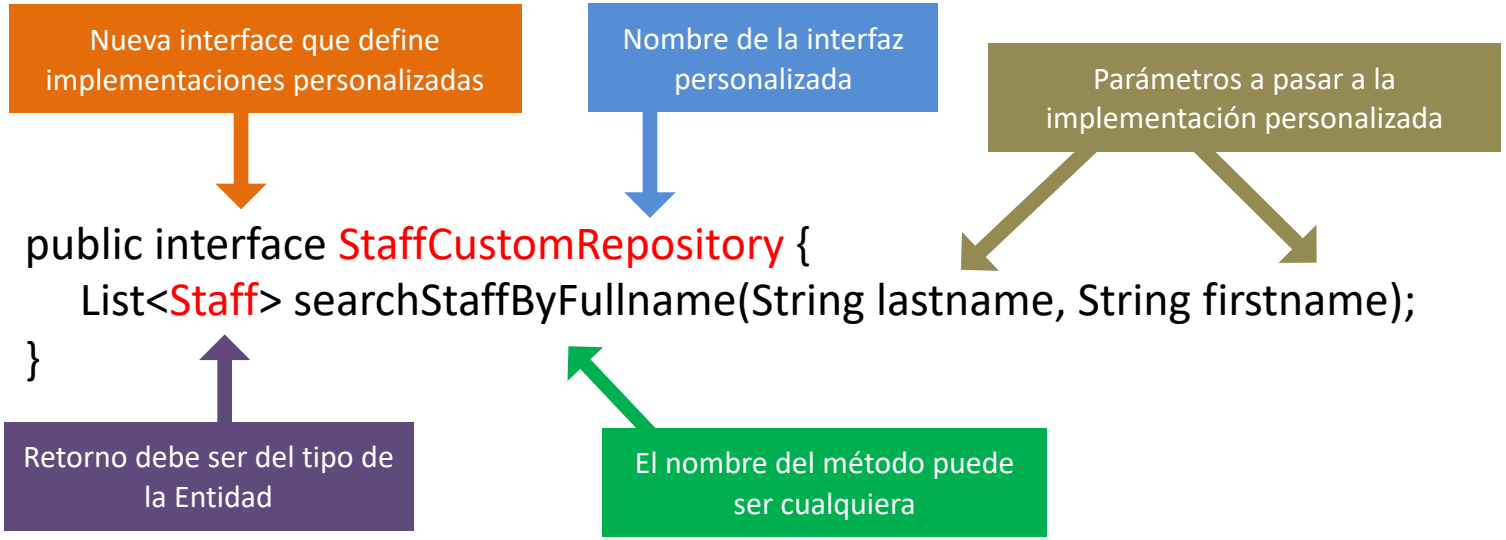
- La clase implementadora de la nueva interfaz definida debe llamarse como la interface que implementa más concatenándole la palabra "**Impl**" (de *Implementation*).

- Ésta clase implementadora puede inyectar el **EntityManager** a través de **@PersistenceContext** para así implementar la consulta a través del API de JPA.

- Por último, la interface *repository* aparte de extender de **CrudRepository**, **PagingAndSortingRepository** o **JpaRepository**, deberá también extender de la nueva interface personalizada definida.

vi.iii Implementaciones personalizadas (c)

- *Custom implementations*. Interfaz personalizada.



Confidencial 2020

vi.iii Implementaciones personalizadas (d)

- *Custom implementations*. Implementación personalizada.

```
public class StaffCustomRepositoryImpl implements StaffCustomRepository {

    @PersistenceContext
    private EntityManager entityManager;

    public List<Staff> searchStaffByFullname(String lastname, String firstname) {
        return entityManager.createQuery(jpql, Staff.class)
            ...
            .getResultList();
        }
}
```

Nombre de la clase Implementadora

Interfaz personalizada

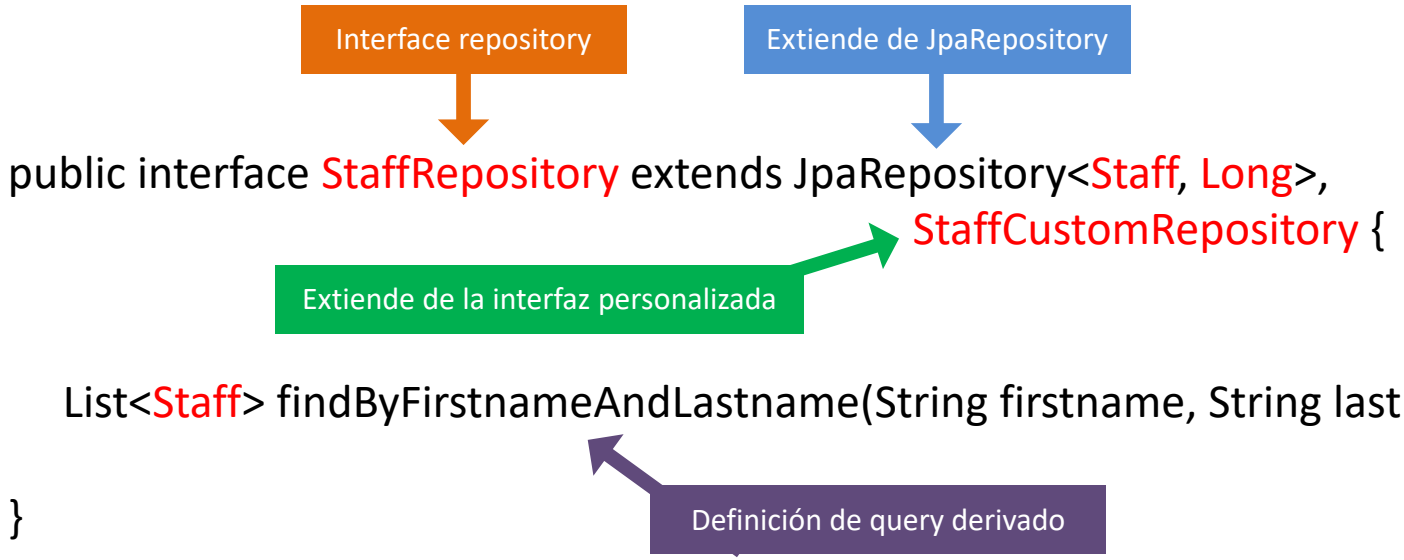
Inyección EntityManager

Implementación personalizada. API JPA

Confidencial 2020

vi.iii Implementaciones personalizadas (e)

- *Custom implementations*. Definición de interface *repository*.



Confidencial 2020

vi.iii Implementación capa de datos con Spring Data JPA

a. Operaciones CRUD
b. Derived Queries
c. Paginación y Ordenamiento
d. @Query Methods
e. Implementaciones personalizadas

Práctica h. Parte 2 – Implementación capa Repository Spring Data JPA

vi.ii Spring Data JPA. Práctica h. (a)

- Práctica h. Parte 2 – Implementación capa Repository Spring Data JPA

- Implementar la capa *repository* de una aplicación Java Empresarial mediante Spring Data JPA.
- Implementar **CrudRepository**, **PagingAndSortingRepository** y **JpaRepository**.
- Implementar *queries* JPA y nativos.
- Implementar *derived queries*.
- Crear implementaciones personalizadas de repositorios JPA.

Esta página fue intencionalmente dejada en blanco.

vi.iv Otros repositorios Spring Data

Objetivos de la lección

vi.iv Otros repositorios Spring Data

- Revisar el módulo de Spring Data MongoDB.
- Comprender la configuración de beans de Spring Data MongoDB.
- Implementar operaciones CRUD con Spring Data MongoDB.
- Revisar el módulo de Spring Data JDBC.
- Comprender la configuración de beans de Spring Data JDBC.
- Implementar operaciones CRUD con Spring Data JDBC.

vi.iv Otros repositorios Spring Data

a. Spring Data MongoDB

Práctica i. Repositorios Spring Data MongoDB

b. Spring Data JDBC

Práctica j. Repositorios Spring Data JDBC

vi.iv Otros repositorios Spring Data (a)

- Spring Data es un proyecto de alto nivel que define un API genérica y abstracta que permite que se incluyan otro tipo de repositorios a la familia de repositorios de Spring Data.

- Spring Data – *One API to rule them all*.

- Otro tipo de repositorios populares son Spring Data MongoDB y el recientemente agregado módulo Spring Data JDBC.

- Spring JDBC no es Spring Data JDBC.

vi.iv Otros repositorios Spring Data

a. Spring Data MongoDB

Práctica i. Repositorios Spring Data MongoDB

b. Spring Data JDBC

Práctica j. Repositorios Spring Data JDBC

vi.iv Spring Data MongoDB (a)

- Como su nombre lo sugiere, Spring Data MongoDB trabaja con el repositorio de datos MongoDB.

- MongoDB persiste la información en base a documentos y esos documentos están definidos en formato JSON (JavaScript Object Notation).

- MongoDB persiste los documentos en colecciones, en lugar de tablas como lo hacen las bases de datos relacionales.

vi.iv Spring Data MongoDB (b)

- Los objetos JSON soportan objetos embebidos así como mantener listas o arreglos de datos en su estructura.

- MongoDB persiste toda la estructura del JSON como un único documento, incluyendo campos embebidos y listas o arreglos de información sin necesidad de agregar relaciones a otras “tablas” o colecciones externas.

vi.iv Spring Data MongoDB (c)

- Spring Data MongoDB utiliza el mismo principio de definición de interfaces *repository* a través del patrón de diseño *Repository Pattern* que define Spring Data a través del módulo Spring Data Commons.

- Spring Data MongoDB define entidades de persistencia basada en documentos en lugar de entidades *Entity* del estándar JPA.

vi.iv Spring Data MongoDB (d)

- Dependencias Spring MongoDB.

- Agregar las dependencia:
  - **org.springframework.data:spring-data-mongodb:${version}**

- Para efectos de testing será necesario configurar un servidor MongoDB embebido, para ello agregar a dependencia:
  - **cz.jirutka.spring:embedmongo-spring:${version}**

vi.iv Spring Data MongoDB (e)

- Configuración Spring Data MongoDB por medio de Java Config.

- Definir beans **MongoClient** y **MongoTemplate** (a):

```
@Bean
@Profile("testing")
public MongoClient mongoClient() throws IOException {
    EmbeddedMongoFactoryBean embeddedMongo = new EmbeddedMongoFactoryBean();
    embeddedMongo.setBindIp("localhost");

    MongoClient mongoClient = embeddedMongo.getObject();
    return mongoClient;
}
```

vi.iv Spring Data MongoDB (f)

- Configuración Spring Data MongoDB por medio de Java Config.

- Definir beans **MongoClient** y **MongoTemplate** (b):

```
@Bean
public MongoClient mongoClient() {
    return new MongoClient("localhost");
}

@Bean
public MongoTemplate mongoTemplate() throws IOException {
    MongoTemplate mongoTemplate = new MongoTemplate(mongoClient(), "testdb");
    return mongoTemplate;
}
```

vi.iv Spring Data MongoDB (g)

- La configuración de **MongoClient** y **MongoTemplate** también puede realizarse a través de configuración por XML, utilizando el *namespace mongo* el cual es habilitado las dependencias Spring Data MongoDB.

```
<mongo:mongo-client id="mongoClient" host="localhost" />

<bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate">
    <constructor-arg ref="mongoClient" />
    <constructor-arg value="testdb" />
</bean>
```

vi.iv Spring Data MongoDB (h)

- El *namespace mongo* provee un *tag* para definir el bean **MongoTemplate** de manera automática a través de un bean **MongoDbFactory**.

```
<mongo:mongo-client id="mongoClient" host="localhost" />

<mongo:db-factory id="mongoDbFactory" mongo-ref="mongoClient" dbname="testdb"/>

<mongo:template  db-factory-ref="mongoDbFactory" />
```

vi.iv Spring Data MongoDB (i)

- Definir los documentos (entidades) a persistir (a):

```
@Data
@Document
public class Department {
    @Id
    private Integer id;
    private String name;

    @DBRef
    private Staff chair;

    public Department(Integer id,
                String name, Staff chair) {
        // set properties
    }
}
```

```
@Data
@Document
public class Staff {
    @Id
    private Integer id;

    private Person member;

    public Staff(Integer id,
                Person member) {
        // set properties
    }
}
```

vi.iv Spring Data MongoDB (j)

- Definir los documentos (entidades) a persistir (b):

- La clase **Person** únicamente se utiliza como *value object*. **Person** no es otro documento.

```
@Data
public class Person {
    private String firstName;
    private String lastName;

    public Person(String firstName,
                  String lastName) {
        // set properties
    }
}
```

vi.iv Spring Data MongoDB (k)

- Similar a los repositorios Spring Data JPA, Spring Data MongoDB ofrece implementación automática de repositorios mediante las interfaces **CrudRepository**, **PagingAndSortingRepository** e integra la interfaz **MongoRepository** la cual es una implementación específica para el motor de persistencia MongoDB.

- Las interfaces *repository* para MongoDB se definen de la misma forma que se especifican los repositorios Spring Data JPA.

vi.iv Spring Data MongoDB (I)

- Para especificar un repositorio MongoDB es necesario declarar una interface que extienda de **CrudRepository**, **PagingAndSortingRepository** o de **MongoRepository** y defina el tipo de documento (entidad) a persistir así como el tipo del identificador del documento.

- Spring Data MongoDB soporta paginación mediante las interfaces **Pageable** y **Page**.

- También, Spring Data MongoDB, soporta ordenamiento mediante la interfaz **Sort**.

vi.iv Spring Data MongoDB (m)

- Spring Data MongoDB soporta la implementación de *derived queries* mediante el análisis de la firma de los métodos especificados en la interface *repository* declarada.

- Por otro lado, Spring Data MongoDB soporta la definición de **@Query** *methods* para definir consultas mediante el lenguaje de consultas propietario de MongoDB.

vi.iv Spring Data MongoDB (n)

- Definición de **StaffRepository**, extensión de **PagingAndSortingRepository**.

```
public interface StaffRepository
            extends PagingAndSortingRepository<Staff, Integer> {

   List<Staff> findByMemberLastName(String lastName);

   @Query("{ 'member.firstName' : ?0 }")
   List<Staff> findByFirstName(String firstName);
}
```

```
@Data
@Document
public class Staff {
   @Id
   private Integer id;
   private Person member;
}
```

vi.iv Spring Data MongoDB (ñ)

- Definición de **DepartmentRepository**, extensión de **MongoRepository**.

```
public interface DepartmentRepository
            extends MongoRepository<Department, String> {

    Optional<Department> findByName(String name);

    @Query("{ 'name' : { $regex: ?0 } }")
    List<Department> findNameByPattern(String pattern);
}
```

```
@Data
@Document
public class Department {
    @Id
    private Integer id;
    private String name;

    @DBRef
    private Staff chair;
}
```

vi.iv Spring Data MongoDB (o)

- Una vez definidas las interfaces *repository* es necesario habilitar los repositorios MongoDB a través de la anotación **@EnableMongoRepositories** en una clase de configuración **@Configuration**.

- La anotación **@EnableMongoRepositories** requiere el nombre del paquete base que contiene las interfaces *repository* de Spring Data MongoDB:
  @EnableMongoRepositories(
  basePackages="package.to.scan.repositories")

vi.iv Spring Data MongoDB (p)

- De forma análoga, es posible habilitar la configuración de repositorios Spring Data MongoDB mediante XML a través de la etiqueta: <**mongo:repositories** base-package="package.to.scan.repositories"/>.

- Una vez definidos los beans **MongoClient**, **MongoTemplate**, definidas las interfaces *repository* que extienden de **CrudRepository**, **PagingAndSortingRepository** o **MongoRepository** y, habilitados los repositorios de Spring Data MongoDB a través de **<mongo:respositories>** o mediante **@EnableMongoRepositories**, es posible empezar a implementar código de acceso a datos a través de Spring Data MongoDB.

vi.iv Spring Data MongoDB (q)

- Opcional, es posible agregar la configuración de Spring Tx, para habilitar transaccionabilidad SQL, a través de la anotación **@EnableTransactionManagement**.

- En caso de habilitar el soporte transaccional de Spring Tx, es necesario registrar un bean de tipo **MongoTransactionManager** el cual esuna implementación de la interface **PlatformTransactionManager**.

- Para habilitar transaccionabilidad en MongoDB se requiere version 4.0+ del servidor **mongod** y que el *cluster* de MongoDB esté configurado en *ReplicaSet*. Fuera de alcance en este curso.

vi.iv Otros repositorios Spring Data

a. Spring Data MongoDB

Práctica i. Repositorios Spring Data MongoDB

b. Spring Data JDBC

Práctica j. Repositorios Spring Data JDBC

vi.iv Otros repositorios Spring Data. Práctica i. (a)

- Práctica i - Repositorios Spring Data MongoDB

- Implementar la capa *repository* de una aplicación Java Empresarial mediante Spring Data MongoDB.
- Implementar **PagingAndSortingRepository** y **MongoRepository**.
- Implementar *queries* específicos del motor de persistencia MongoDB.
- Implementar *derived queries*.

vi.iv Otros repositorios Spring Data

a. Spring Data MongoDB

Práctica i. Repositorios Spring Data MongoDB

b. Spring Data JDBC

Práctica j. Repositorios Spring Data JDBC

vi.iv Spring Data JDBC (a)

- Similar a Spring Data JPA, Spring Data JDBC trabaja con bases de datos relacionales SQL a través de JDBC.

- Spring Data JDBC es una evolución de Spring JDBC el cual define interfaces y funciones *callback* para implementar operaciones JDBC a través de *templates*.

- Spring JDBC ha sido muy utilizado y es altamente recomendable cuando no se tiene una estructura de tablas compleja.

vi.iv Spring Data JDBC (b)

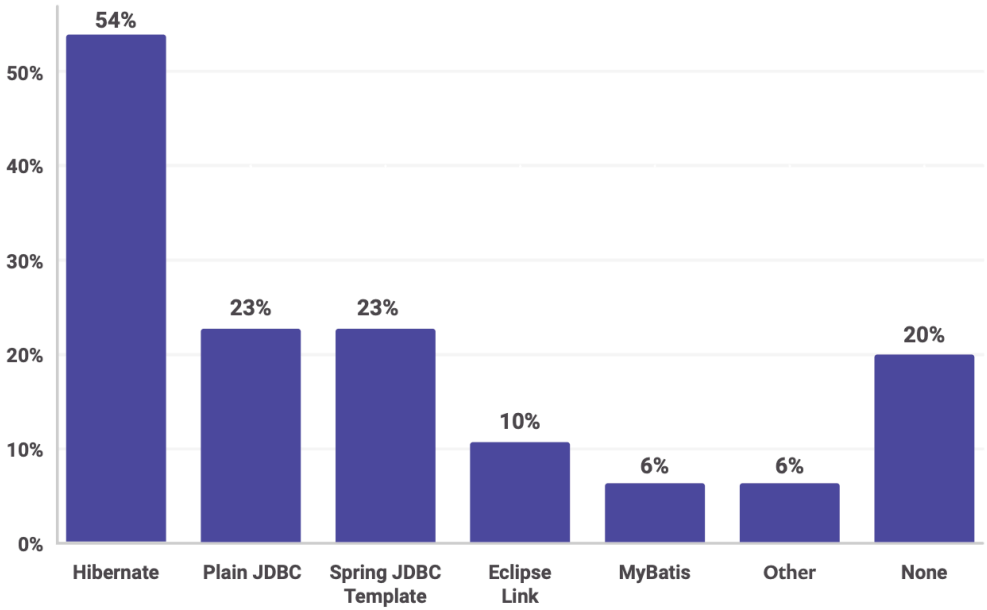
- JPA ofrece muchos beneficios como carga de entidades "*lazy*" o "*eager*", cache de 1er y 2do nivel, seguimiento de consultas hasta que sean persistidas ("*commited*") en la base de datos, etc.

- En muchos proyectos Java Empresariales, no se requiere la complejidad de JPA para implementar acceso a datos.

- Spring Data JDBC es un nuevo mecanismo de consultas SQL sin la complejidad de JPA.

vi.iv Spring Data JDBC (c)

- Encuesta de uso de Frameworks ORM 2018.



Confidencial 2020

vi.iv Spring Data JDBC (d)

- Spring Data JDBC apunta a ser un modelo de consultas mucho más simple que JPA, para requerimientos menos complejos y que no requieran las capacidades del estándar JPA.

vi.iv Spring Data JDBC (d)

- Spring Data JDBC apunta a ser un modelo de consultas mucho más simple que JPA, para requerimientos menos complejos y que no requieran las capacidades del estándar JPA.

Confidencial 2020

vi.iv Spring Data JDBC (e)

- Spring Data JDBC utiliza el mismo modelo de programación definido en Spring Data Commons es decir:
  - Se deben definir y mapear entidades a través de anotaciones.
  - Se debe habilitar el escaneo de repositorios Spring Data JDBC a través de la anotación **@EnableJdbcRepositories**.
  - Se deben definir interfaces *repository* que extiendan de **CrudRepository**.

Confidencial 2020

vi.iv Spring Data JDBC (f)

- Spring Data JDBC permite definir consultas SQL (nativas) a través de la anotación **@Query** sobre métodos definidos en las interfaces *repository*, es decir soporta ***@Query** methods*.

- Spring Data JDBC únicamente implementa **CrudRepository**.

- A la fecha del *release* Spring Data JDBC 1.x no soporta:
  - Definir interfaces *repository* que extiendan de la interfaz **PagingAndSortingRepository**.
  - *Derived queries*.

vi.iv Spring Data JDBC (g)

- Spring Data JDBC no ofrece generación de esquemas automáticos como Spring Data JPA (a través de Hibernate), lo cual requiere que el desarrollador defina el esquema de base de datos.
  - Sin embargo no es realmente un problema debido a que, ningún proyecto productivo tiene habilitada la opción de generación automática de esquema de Hibernate("*create-drop*").

- Spring Data JDBC no maneja sesiones, no maneja *dirty checks*, no maneja cache, no maneja alguna complejidad extra más que ejecutar operaciones de lectura o escritura en el momento en que dichos métodos sean invocados.

vi.iv Spring Data JDBC (h)

- Spring Data JDBC cambia el paradigma al que ”nos acostumbró” JPA y Hibernate, por tanto:
  - Si se modifica una entidad que fue recuperada (leída) por el método **findOne()** o **findAll()** de Spring Data JDBC y no se invoca explícitamente al método **save()**, ¡no ocurre nada!

- Spring Data JDBC es la forma más fácil y simple de implementar repositorios JDBC sin embargo, tiene una curva de aprendizaje debido al conocimiento previo en el manejo de entidades y acceso a datos con el estándar JPA.

vi.iv Spring Data JDBC (i)

- Dependencias Spring Data JDBC.

- Agregar la dependencia:
  - **org.springframework.data:spring-data-jdbc:${version}**

- Para efectos de testing se necesitará configurar una base de datos en memoria H2, para ello agregar la dependencia:
  - **com.h2database:h2:${version}**

vi.iv Spring Data JDBC (j)

- Configuración Spring Data JDBC por medio de Java Config.

- Definir beans **DataSource** y **NamedParameterJdbcTemplate** (a):

```
@Bean
public NamedParameterJdbcOperations operations(DataSource dataSource) {
    return new NamedParameterJdbcTemplate(dataSource);
}

@Bean
public DataSource dataSource() {
    BasicDataSource ds = new BasicDataSource();
    // Datasource configuration
    return ds;
}
```

org.apache.commons.dbcp.BasicDataSource

Confidencial 2020

vi.iv Spring Data JDBC (k)

- Configuración Spring Data JDBC por medio de Java Config.

- Definir beans **DataSource** y **NamedParameterJdbcTemplate** (b):

```
@Bean
@Profile("testing")
public DataSource dataSource() {
    return new EmbeddedDatabaseBuilder()
                        .generateUniqueName(true)
                        .setType(EmbeddedDatabaseType.H2)
                        .addScript("db/jdbc/schema.sql")
                        .addScript("db/jdbc/data.sql")
                        .build();
}
```

vi.iv Spring Data JDBC (I)

- Definir las entidades JDBC a persistir (a):

```
@Data
@Table
public class Department {

    @Id
    private Integer id;
    private String name;
    private Chair chair;

    public Department(String name,
                      Chair chair) {
        // set properties
    }
}
```

org.springframework.data.relational.core.mapping.Table

org.springframework.data.annotation.Id

```
@Data
@Table
public class Chair {

    @Id
    private Integer department;
    private String name;

    public Chair(String name) {
        // set properties
    }
}
```

vi.iv Spring Data JDBC (m)

- Es posible utilizar también la anotación **@Column**, del paquete **org.springframework.data.relational.core.mapping**, para definir el nombre de una columna.

- La anotación **@Column** mapea el nombre de una propiedad a una columna con el mismo nombre de la propiedad, sustituyendo la composición de nombres *camel-case* a *snake-case*.

- La anotación **@Table** mapea el nombre de la clase a una tabla definida, con el mismo nombre, sustituyendo la composición de nombres *camel-case* a *snake-case*.

vi.iv Spring Data JDBC (n)

- Similar a los repositorios Spring Data JPA, Spring Data JDBC ofrece implementación automática de repositorios, únicamente a través de la interfaz **CrudRepository**.

- Para definir una interfaz *repository* para Spring Data JDBC, es necesario definir una interface y extender de **CrudRepository**, a su vez es requerido definir el tipo de la entidad y el tipo del identificador de la entidad Spring Data JDBC.

vi.iv Spring Data JDBC (ñ)

- Definición de **DepartmentRepository**, extensión de **CrudRepository**.

```
public interface DepartmentRepository extends CrudRepository<Department, Integer> {

    @Query("SELECT " +
            "DEPARTMENT.id AS id, DEPARTMENT.name AS name, " +
            "CHAIR.department AS chair_department, CHAIR.name AS chair_name " +
            "FROM DEPARTMENT LEFT OUTER JOIN CHAIR AS CHAIR ON " +
            "CHAIR.DEPARTMENT = DEPARTMENT.id " +
            "WHERE DEPARTMENT.name =:name")
    Optional<Department> findByName(@Param("name") String name);
}
```

vi.iv Spring Data JDBC (o)

- Por el momento Spring Data JDBC no soporta paginación ni ordenamiento, a través de la interface **PagingAndSortingRepository**.

- Spring Data JDBC no soporta paginación ni ordenamiento mediante las interfaces **Pageable**, **Page** y **Sort**.

- Spring Data JDBC no soporta *derived queries*.

- A la fecha, el *release* Spring Data JDBC 1.x no incluye soporte para configuración por XML.

vi.iv Spring Data JDBC (p)

- Una vez definidas las interfaces *repository* es necesario habilitar los repositorios Spring Data JDBC a través de la anotación **@EnableJdbcRepositories** en una clase de configuración **@Configuration**.

- La anotación **@EnableJdbcRepositories** requiere el nombre del paquete base que contiene las interfaces *repository* definidas de Spring Data JDBC:
  @EnableJdbcRepositories(
  basePackages="package.to.scan.repositories")

vi.iv Spring Data JDBC (q)

- Opcional, es posible agregar la configuración de Spring Tx, para habilitar transaccionabilidad SQL, a través de la anotación **@EnableTransactionManagement**.

- En caso de habilitar el soporte transaccional de Spring Tx, es necesario registrar un bean de tipo **DataSourceTransactionManager** el cual es una implementación de la interface **PlatformTransactionManager**.

vi.iv Otros repositorios Spring Data

a. Spring Data MongoDB

Práctica i. Repositorios Spring Data MongoDB

b. Spring Data JDBC

Práctica j. Repositorios Spring Data JDBC

vi.iv Otros repositorios Spring Data. Práctica j. (a)

- Práctica j. Repositorios Spring Data JDBC

- Implementar la capa *repository* de una aplicación Java Empresarial mediante Spring Data JDBC.
- Implementar **CrudRepository**.
- Implementar *queries* a través de lenguaje SQL nativo.

Resumen de la lección

vi.iv Otros repositorios Spring Data

- Revisamos el módulo de Spring Data MongoDB.
- Comprendimos a configurar los beans requeridos para habilitar repositorios Spring Data MongoDB.
- Implementamos operaciones CRUD con Spring Data MongoDB.
- Revisamos el módulo de Spring Data JDBC.
- Comprendimos a configurar los beans requeridos para habilitar repositorios Spring Data JDBC.
- Implementamos operaciones CRUD con Spring Data JDBC.

Esta página fue intencionalmente dejada en blanco.