Java Persistence Api (JPA) con Spring e Hibernate

Universidad Centroamericana "José Simeón Cañas"

Ciclo 01-2020

Operaciones sobre bases de datos son cruciales para una aplicación empresarial

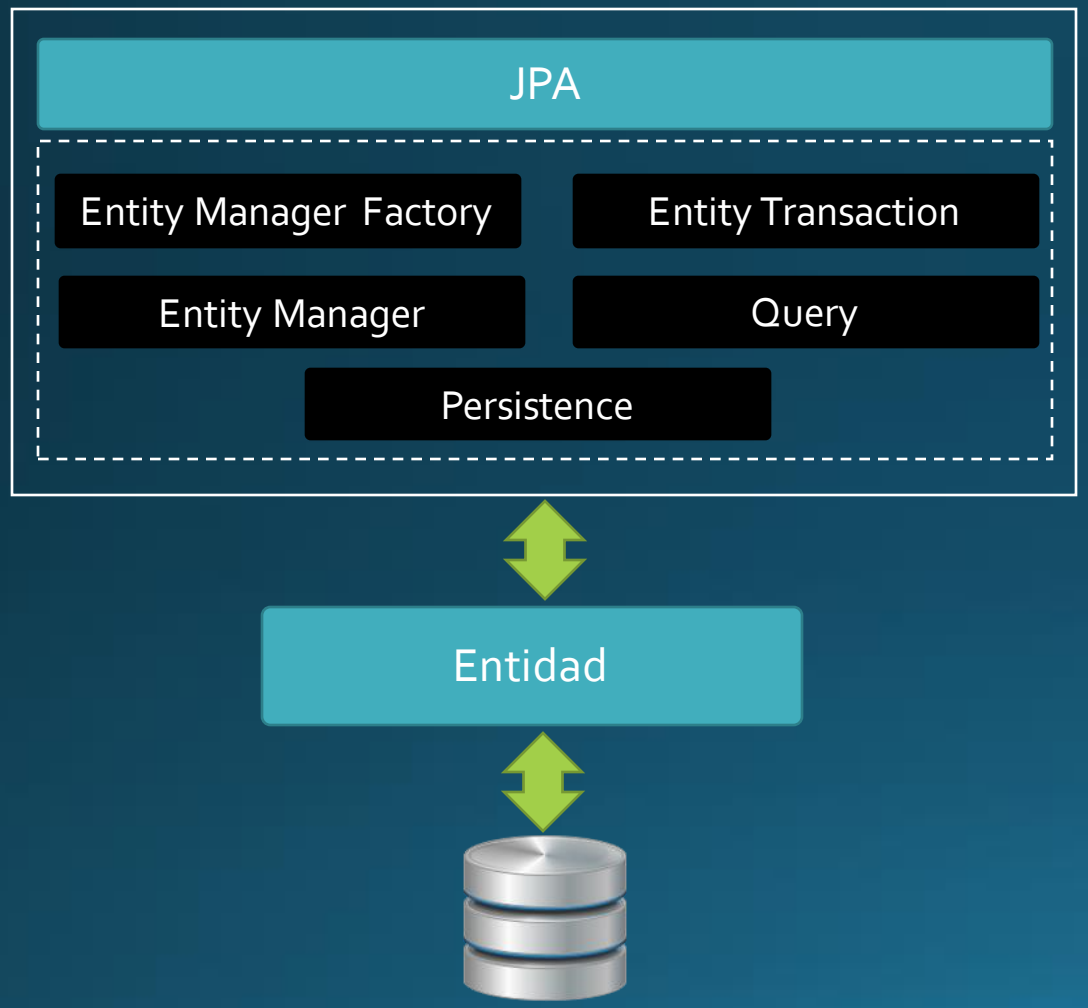
- Para un desarrollador trabajar con base de datos implica la creación de código encargado de manipularla
- Este desarrollo implica tiempo y esfuerzo, y esto se traduce en costos mas grandes para la empresa
- JPA es una API que no es permite disminuir el tiempo de desarrollo de dicho código

Debido a esto, es importante saber utilizar una herramienta que nos permita desarrollar una capa de acceso a datos de forma eficiente, de manera de poder utilizar lo mejor posible los recursos de un proyecto

JPA es una API desarrollada para la Java EE

- Permite operar datos que son nativamente relacionales (bases de datos) en aplicaciones Java, tanto SE como EE
- Esta especificada en la JSR 220, junto con la especificación de la Enterprise Java Beans
- JPA sigue un patrón ORM (Object Relational Mapping), muy similar a otros frameworks como Entity Framework de .NET
- Hay varias implementaciones de JPA (Proveedores):
  - Hibernate
  - TopLink
  - EclipseLink
  - OpenJPA
  - Etc.

Componentes de JPA



- **EntityManager:** Es una interfaz encargada de manejar las operaciones de persistencia de los objetos
- **EntityManagerFactory**: Es el encargado de administrar y crear los objetos EntityManager, gestiona la conexión a la base de datos
- **Entidad:** Son los objetos de persistencia, es cada registro que se encuentra en una tabla de una base de datos
- **EntityTransaction:** Es una interfaz encargada de manejar las transacciones de las operaciones de persistencia
- **Persistence:** Es una clase que permite obtener los objetos **EntityManagerFactory**
- **Query:** Es una interfaz que permite hacer consultas en base a distintos criterios

Proveedores de JPA

- Para poder utilizar JPA es necesario hacerlo a través de un proveedor
- JPA en sí es solamente la especificación
- Utilizaremos Hibernate como proveedor JPA

Un poco de historia de Hibernate…

- Hibernate es una herramienta ORM (Object Relational Mapping) utilizada para persistir y consultar objetos (entidades) desde y hacia una base de datos
- Fue creada por Gavin King junto a otros programadores, para solucionar el problema del manejo de datos relacionales en una aplicación empresarial

Configuración de JPA e Hibernate

- Agregaremos dos nuevas dependencia a nuestro archivo pom.xml
- Estas dependencia contienen las clases necesarias para trabajar con JPA utilizando Hibernate, Spring y PostgreSQL

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```

Dependencias necesarias

- **Spring-boot-starter-data-jpa:** Contiene las clases necesarias para implementar la especificación JPA con las funcionalidades propias de Hibernate
- **Postgresql:** Contiene las clases necesarias para la conectividad JDBC hacia una base de datos Postgres

Creación de clase de configuración de Hibernate (JpaConfiguration.java)

- Crearemos una clase, similar a la de Spring, que contendrá los beans de configuración de Hibernate. A continuación se detallan los métodos que tendrá dicha clase

```
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
    LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();
    em.setDataSource(dataSource());
    em.setPersistenceUnitName("modelo-persistence");
    em.setPackagesToScan("com.uca.capas.modelo.domain");

    JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    em.setJpaVendorAdapter(vendorAdapter);
    em.setJpaProperties(hibernateProperties());

    return em;
}
```

JpaConfiguration.java (cont.)

```java
@Bean
public DataSource dataSource(){
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("org.postgresql.Driver");
    dataSource.setUrl("jdbc:postgresql://192.168.1.13:5432/ventasng");
    dataSource.setUsername("postgres");
    dataSource.setPassword("postgres");

    return dataSource;
}

Properties hibernateProperties() {
    Properties properties = new Properties();
    properties.setProperty("hibernate.show_sql", "true");
    properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect");
    properties.setProperty("hibernate.enable_lazy_load_no_trans","true");
    return properties;
}
```

Componentes para la configuración de Hibernate con Spring

- **LocalContainerEntityManagerFactoryBean:** Es utilizado para crear un EntityManagerFactory, esta es creada como un Singleton (una sola instancia compartida para todas las clases), ya que es un objeto muy pesado en características.
- **DataSource:** Contiene la información de conexión a la base de datos que utilizará el Entity Manager
- **Properties:** Son características propias de Hibernate y son utilizadas para depurar queries, operaciones sobre la base de datos, etc.

Propiedades de Hibernate

- **Hibernate.show_sql=true:** Esta propiedad indica si queremos que las consultas SQL que se hacen desde nuestro aplicativo a la base de datos se impriman en la consola con fines de depuración.
- **Hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect:** Hibernate trabaja con el concepto de dialectos, los cuales utiliza para generar el SQL correspondiente a la base de datos que estamos trabajando. En nuestro caso le definiremos el de Postgres, sin embargo tiene soporte para muchos mas, como Oracle, SQL Server, MySQL, etc.
- **Hibernate.enable_lazy_load_no_trans=true:** Esta propiedad sirve para manejar el estado **Lazy** de las relaciones de las entidades (el cual veremos a mayor detalle en las próximas clases)

Ya hemos configurado