**ORM (Object Relational Mapping)**

El Mapeo Objeto-Relacional (ORM, por sus siglas en inglés) es una técnica que facilita la recuperación y persistencia de objetos en bases de datos relacionales. Los motores de ORM permiten a los desarrolladores trabajar con un modelo de objetos y abstraer las interacciones con la base de datos, lo que les permite centrarse más en la lógica de la aplicación en lugar de escribir consultas SQL directamente. Esto se logra a través de la traducción automática de los modelos de objetos en consultas SQL y viceversa.

**Beneficios de ORM en Bases de Datos Grandes**

1. **Reducción del Código SQL**: Permite a los desarrolladores escribir menos líneas de código SQL, lo que hace que el desarrollo sea más rápido y menos propenso a errores de sintaxis.
2. **Mejora de la Seguridad**: Disminuye el riesgo de inyecciones SQL, ya que gestiona de manera adecuada la creación de consultas y la manipulación de datos.
3. **Facilidad de Mantenimiento**: Los modelos de datos se manejan como objetos, lo que puede facilitar la lectura y el mantenimiento del código a largo plazo.
4. **Optimización del Rendimiento**: Aunque puede haber una penalización de rendimiento inicial, el uso de cachés (como la caché de primer nivel en Hibernate) puede mitigar estos efectos en las lecturas sucesivas.
5. **Flexibilidad en la Manipulación de Datos**: Los motores de ORM permiten realizar operaciones complejas, como la carga diferida (lazy loading) y la carga ansiosa (eager loading), lo que puede mejorar la eficiencia en la recuperación de datos.
6. **Adaptabilidad a Cambios en el Modelo de Datos**: Los ORM facilitan la adaptación del código a los cambios en la estructura de la base de datos sin necesidad de reescribir completamente las consultas SQL.
7. **Facilitación de la Programación Orientada a Objetos**: Ayuda a los desarrolladores a trabajar en un entorno que integra la programación orientada a objetos con el almacenamiento de datos relacionales, lo que puede simplificar el desarrollo.
8. **Soporte para Transacciones**: Muchos ORM manejan automáticamente las transacciones, lo que puede reducir la complejidad del manejo de errores y asegurar la consistencia de los datos.
9. **Compatibilidad con Múltiples Bases de Datos**: Permiten a los desarrolladores trabajar con diferentes sistemas de bases de datos sin necesidad de cambiar el código de la aplicación.
10. **Desarrollo Más Rápido**: Al abstraer gran parte de la complejidad de las interacciones con la base de datos, los ORM pueden ayudar a acelerar el tiempo de desarrollo y a reducir la carga cognitiva sobre los desarrolladores.

**JPA (Java Persistence Api)**

JPA, o Java Persistence API, es una interfaz de programación en Java que permite a los desarrolladores gestionar la persistencia de datos en bases de datos relacionales. Su principal función es facilitar la traducción entre el modelo de objetos de un sistema y su capa de persistencia, es decir, las entidades y atributos de una base de datos relacional. Esto incluye la generación y validación de consultas SQL necesarias para guardar y recuperar datos de objetos.

**Funciones principales de JPA**

1. **Gestión de la Persistencia**: Proporciona un conjunto de métodos para interactuar con la base de datos a través de una unidad de persistencia, como el EntityManager.
2. **Operaciones CRUD Simplificadas**: Permite a los desarrolladores enfocarse en la lógica de negocio en lugar de escribir código repetitivo para operaciones de creación, lectura, actualización y eliminación (CRUD).
3. **Consultas Parametrizadas**: JPA permite construir consultas utilizando SQL, HQL (Hibernate Query Language) o lenguajes específicos de dominio, lo que mejora la legibilidad y la seguridad del código.
4. **Manejo de Caché**: Actúa como un área de almacenamiento en caché para minimizar las llamadas a la base de datos, mejorando así el rendimiento de la aplicación.
5. **Validación de Consultas**: Las consultas SQL generadas son validadas en memoria, lo que ayuda a evitar errores de sintaxis y problemas de inyección SQL.
6. **Configuración Flexible**: JPA permite personalizar el comportamiento de la persistencia a través de anotaciones y archivos XML.
7. **Cargas Perezosas y Ágiles**: Puede gestionar la carga de datos de forma perezosa (lazy loading) o ágil (eager loading), lo que permite optimizar el rendimiento según sea necesario.
8. **Compatibilidad con Diferentes ORM**: Aunque se usa comúnmente con Hibernate, JPA es compatible con otros motores de mapeo objeto-relacional (ORM) como EclipseLink y OpenJPA.

**Desafíos de JPA**

1. **Complejidad**: A pesar de su facilidad de uso, los motores ORM como JPA pueden ser complejos y requieren una comprensión profunda de sus funcionalidades y limitaciones.
2. **Impedancia de Desajuste**: Existe una discrepancia entre los modelos orientados a objetos y las bases de datos relacionales, lo que puede complicar el mapeo de datos.
3. **Dependencias**: La inclusión de bibliotecas de JPA puede aumentar el tamaño del proyecto debido a las múltiples dependencias necesarias.
4. **Rendimiento**: La configuración incorrecta de JPA puede llevar a problemas de rendimiento, lo que requiere un perfilado cuidadoso de la aplicación.

**Spring Boot**

Para hablar de Spring Boot primero es necesario abordar su antecesor que es Spring Framework. **Spring Framework** es un marco de trabajo para el desarrollo de aplicaciones en Java, diseñado para proporcionar soporte integral a nivel de infraestructura. Permite a los desarrolladores enfocarse en la lógica de la aplicación en lugar de los detalles técnicos de la infraestructura. Spring facilita la creación de aplicaciones utilizando "Plain Old Java Objects" (POJOs) y ofrece la posibilidad de aplicar servicios empresariales de manera no invasiva.

**Historia y Problemas Previos al Desarrollo de Spring Framework**

Antes de la aparición de Spring, el desarrollo de aplicaciones Java, especialmente a nivel empresarial, presentaba varios desafíos significativos:

1. **Complejidad en la Gestión de Transacciones**: Los desarrolladores necesitaban manejar manualmente las transacciones utilizando APIs específicas, lo que resultaba en código complejo y difícil de mantener.
2. **Falta de Flexibilidad en la Inyección de Dependencias**: Las aplicaciones Java estaban altamente acopladas, lo que dificultaba el testeo y el mantenimiento. El uso de patrones como Factory o Service Locator ayudaba, pero no eliminaba completamente el problema del acoplamiento.
3. **Monolitismo en las Aplicaciones Empresariales**: Las soluciones existentes para desarrollar aplicaciones empresariales, como EJB (Enterprise JavaBeans), eran pesadas y obligaban a los desarrolladores a seguir enfoques intrusivos, lo que resultaba en aplicaciones monolíticas y difíciles de escalar.

**Surgimiento de Spring**

Spring fue desarrollado para abordar estos problemas al proporcionar un marco ligero y modular que permite a los desarrolladores usar solo los componentes que necesitan, sin imponer una arquitectura monolítica. Las características principales de Spring incluyen:

* **Inversión de Control (IoC)** y **Inyección de Dependencias (DI)**: Spring utiliza IoC para gestionar la creación y el ciclo de vida de los objetos, permitiendo un bajo acoplamiento entre los componentes. Esto facilita el testeo y el mantenimiento de la aplicación.
* **Modularidad**: Spring está compuesto por varios módulos, como el Core Container, el módulo de acceso a datos (JDBC, ORM, etc.), el módulo web (Spring MVC), y más, permitiendo a los desarrolladores elegir y utilizar solo los componentes necesarios.
* **Soporte para Programación AOP (Aspect-Oriented Programming)**: Permite la separación de preocupaciones transversales como la seguridad, las transacciones, y el logging, mejorando la mantenibilidad del código.
* **Gestión de Transacciones Declarativas**: Spring permite manejar las transacciones de manera declarativa sin necesidad de escribir código específico para ello, reduciendo la complejidad.

Aunque el desarrollo de Spring Framework supero estos desafíos también trajo consigo problemas que se van a abordar más adelante cuando se hable del desarrollo de spring boot como tal que es el tema principal. Spring Boot es un marco de trabajo en Java que facilita la creación de aplicaciones web y servicios RESTful. Su desarrollo se basa en el marco Spring, que ya existía pero presentaba ciertos problemas que Spring Boot busca resolver. A continuación, se detallan algunos de estos problemas:

**Problemas antes de Spring Boot**

1. **Configuración Compleja**: Las aplicaciones en Spring tradicional requerían una configuración extensa, lo que podía llevar mucho tiempo.
2. **Dependencias**: La gestión de dependencias era complicada, ya que se necesitaban múltiples bibliotecas que debían ser configuradas manualmente.
3. **Despliegue**: El proceso de despliegue de aplicaciones era engorroso y requería mucha intervención manual.
4. **Escalabilidad**: Las aplicaciones no siempre eran fáciles de escalar debido a sus configuraciones rígidas.
5. **Pruebas**: La integración de pruebas dentro del ciclo de desarrollo no era tan fluida como se quería.
6. **Estandarización**: La falta de convenciones llevó a una variabilidad en la calidad y estructura de las aplicaciones.
7. **Performance**: Las aplicaciones podían ser lentas si no se configuraban adecuadamente, debido a la sobrecarga de recursos en configuraciones innecesarias.
8. **Reusabilidad**: La reutilización de código y componentes era limitada, lo que resultaba en más duplicación de esfuerzos.

**Resumen de la Historia de Spring Boot**

Spring Boot fue creado para simplificar el desarrollo de aplicaciones y abordar las deficiencias del marco Spring tradicional. A partir de su lanzamiento, ha ganado popularidad rápidamente debido a su enfoque en la "convención sobre configuración", lo que permite a los desarrolladores crear aplicaciones de manera más ágil y eficiente. Algunas características clave incluyen:

1. **Configuración Automática**: Detecta automáticamente las configuraciones necesarias basadas en las bibliotecas presentes en el proyecto.
2. **Standalone**: Permite crear aplicaciones independientes que pueden ejecutarse sin un servidor externo.
3. **Microservicios**: Facilita la creación de aplicaciones basadas en microservicios con una estructura ligera y escalable.
4. **Integración con Spring Framework**: Se basa en la sólida infraestructura de Spring, ofreciendo todas sus capacidades sin la complejidad adicional.
5. **Facilidad de Despliegue**: Proporciona herramientas para empaquetar y desplegar aplicaciones de manera sencilla.
6. **Métricas y Monitorización**: Incluye funcionalidades para monitorear aplicaciones y recopilar métricas de rendimiento.
7. **Soporte para Pruebas**: Mejora el soporte para pruebas unitarias y de integración.
8. **Comunidad Activa**: Respaldado por una comunidad amplia, lo que facilita el acceso a recursos y soporte.

Spring Boot ha revolucionado la forma en que se desarrollan aplicaciones Java, al ofrecer una solución más eficiente y menos propensa a errores en comparación con sus predecesores 2.

**Bibliografía**

G. Vial, "Lessons in Persisting Object Data Using Object-Relational Mapping" in IEEE Software, vol. 36, no. 06, pp. 43-52, 2019.  
doi: 10.1109/MS.2018.227105428. Tomado de: <https://csdl-downloads.ieeecomputer.org/mags/so/2019/06/08356175.pdf?Expires=1725396723&Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc291cmNlIjoiaHR0cHM6Ly9jc2RsLWRvd25sb2Fkcy5pZWVlY29tcHV0ZXIub3JnL21hZ3Mvc28vMjAxOS8wNi8wODM1NjE3NS5wZGYiLCJDb25kaXRpb24iOnsiRGF0ZUxlc3NUaGFuIjp7IkFXUzpFcG9jaFRpbWUiOjE3MjUzOTY3MjN9fX1dfQ__&Signature=k1oDFNTWBUFbDw4W6ZCxV6NqU~tQhuh6SoNCTinEM-oCm2sx~c1Tt8XTyZpZnjJsJ-awLZCIoOR6C56D-M65sBVRP9j9PuDqaSSZngXXGCBgjb8hormjZVR7uj439ktNArdXKtEc~0dLUhoy-CScfJudqzJRlb5aKxBFiLvyAmPKzO6IzQnJ6PHUNsyNtKo3PsE~0nYTFO0NHFz3VdZttgPbk1ubtyN9sOb4YQM5sWfHUqFuSU8pDDFtj1ax~YQYKZVT9Jy6CW4ENBN32hl~c5fFX0U3-1OsZuxQmcixsV35ZvnePNahoQiHNEJBrX-oF~63in376iz~gaq01j8iQQ__&Key-Pair-Id=K12PMWTCQBDMDT>

Dhalla, H. K. (2021, June). A performance comparison of restful applications implemented in Spring Boot Java and MS. NET Core. In *Journal of Physics: Conference Series* (Vol. 1933, No. 1, p. 012041). IOP Publishing. Tomado de: <https://iopscience.iop.org/article/10.1088/1742-6596/1933/1/012041/pdf>

Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R., Risberg, T., ... & Webb, P. (2004). The spring framework-reference documentation. *interface*, *21*, 27. Tomado de: <https://docs.spring.io/spring-framework/docs/3.2.17.RELEASE/spring-framework-reference/pdf/spring-framework-reference.pdf>