**¿Qué es Spring Data?**

Spring Data es una parte esencial del ecosistema Spring. Es un conjunto de proyectos que facilita el acceso y manejo de datos en diversas bases de datos utilizando el modelo de Spring. El propósito de Spring Data es simplificar el desarrollo de aplicaciones al proporcionar una abstracción sobre las bases de datos, permitiendo a los desarrolladores concentrarse en los problemas de negocio.

**¿Cuáles son los subproyectos de Spring Data?**

Dentro de Spring Data, existen varios subproyectos que están diseñados para trabajar con diferentes tecnologías de bases de datos:

**Spring Data JPA**: Se enfoca en la implementación del proveedor JPA para ORM, lo cual es crucial para manejar bases de datos relacionales.

**Spring Data JDBC:** Proporciona un acceso más directo y ligero a las bases de datos relacionales.

**Spring Data MongoDB**: Ofrece soporte para las operaciones CRUD sobre bases de datos basadas en documentos como MongoDB.

**Spring Data Cassandra**: Diseñado para mapas de columnas distribuidas, típico en bases de datos NoSQL como Cassandra.

Cada subproyecto está dirigido a un tipo específico de base de datos, ofreciendo herramientas especializadas para operar de manera efectiva con ellas.

¿Cómo conectar el proyecto a la base de datos?

Para conectar nuestro proyecto a una base de datos MySQL desde IntelliJ IDEA, debemos editar el archivo application.properties dentro del directorio resources. Aquí se deben poner cuatro instrucciones esenciales para establecer la conexión:

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/fixeria?createDatabaseIfNotExist=true

spring.datasource.username=root

spring.datasource.password=admin

**¿Qué parámetros son esenciales en application.properties?**

spring.datasource.driver-class-name: Especifica el tipo de base de datos.

spring.datasource.url: Dirección y nombre de la base de datos junto con la instrucción createDatabaseIfNotExist=true para MySQL.

spring.datasource.username y password: Credenciales de acceso a la base de datos.

**¿Para qué sirve Hikari CP?**

Hikari CP es un gestor automático de conexiones a base de datos, optimizando la cantidad de conexiones según la concurrencia de usuarios en la aplicación. Ofrece un mejor rendimiento y es fundamental para la eficiencia y escalabilidad del sistema.

**¿Cómo configurar Spring JPA?**

Para gestionar dinámicamente el esquema de base de datos, podemos añadir y configurar Spring JPA:

spring.jpa.hibernate.ddl-auto=update

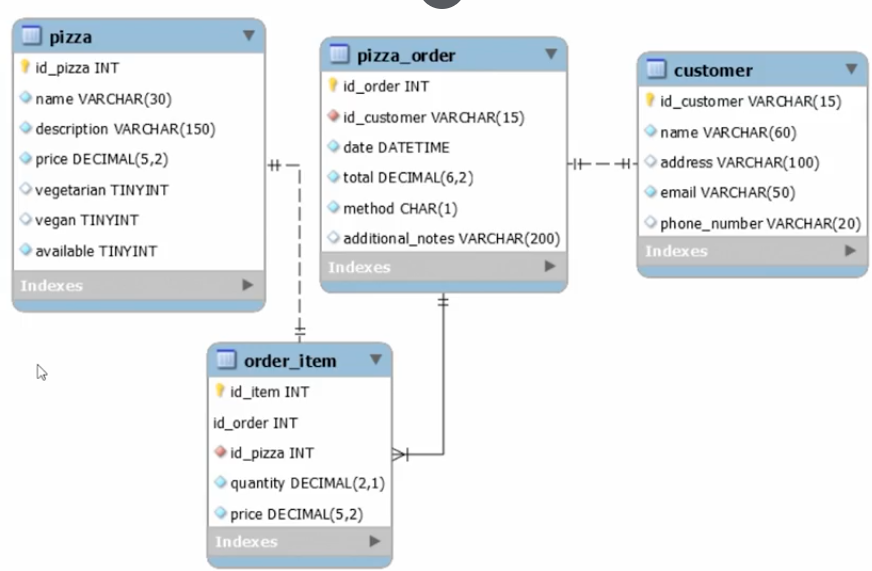
spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto:

update: Se adapta al esquema al sincronizar cambios sin perder datos.

spring.jpa.show-sql: Permite visualizar en la consola las acciones SQL realizadas por nuestra aplicación.

Estas configuraciones permiten que la estructura de las bases de datos se actualice automáticamente a partir de las entidades definidas en nuestro proyecto. Además, habilitar show-sql nos ayuda a verificar y analizar cómo nuestras acciones afectan a la base de datos en tiempo real.



**¿Cuáles son los tipos de relaciones más comunes?**

Uno a Uno: Se representa mediante @OneToOne.

Uno a Muchos: Se logra con @OneToMany.

Muchos a Uno: Utiliza @ManyToOne.

**¿Cómo se trabaja con claves primarias compuestas?**

Creación de claves compuestas es otra parte vital del trabajo con JPA, donde dos o más atributos forman una clave primaria:

Utilizar la anotación @IdClass es esencial para gestionar estas claves compuestas.

**¿Cómo se implementan las anotaciones para asegurar integridad referencial?**

En la integración de estas relaciones, la integridad y el rendimiento son cruciales. Utilizar @JoinColumn es importante. Se usa para especificar la columna a través de la cual están vinculadas dos tablas

**¿Cuáles son las recomendaciones para el uso de estas relaciones?**

Al crear relaciones, es clave tener en cuenta el impacto en el desempeño de la aplicación. No siempre es necesario crear relaciones bidireccionales o añadir todas las posibles relaciones que se puedan modelar.

Simplificar consultas: Mediante consultas {query} o repositorios de Spring, se pueden derivar datos sin necesidad de establecer relaciones directas en todos los contextos.

Evitar sobrecarga: Solo se deben crear las relaciones que sean necesarias estrictamente para garantizar el rendimiento óptimo y la reducción de carga.

**¿Cómo realizar consultas con JDBC Template?**

El JDBC Template es una poderosa herramienta que permite realizar consultas SQL y mapear los resultados a clases Java. Este enfoque nos brinda flexibilidad y control sobre las interacciones con la base de datos.

**¿Cómo facilitan los Spring Data Repositories la interacción con bases de datos?**

Los Spring Data Repositories son elementos fundamentales para desarrollar aplicaciones en Spring, pues simplifican notablemente el manejo de bases de datos al proporcionar operaciones comunes necesarias. Esta eficiencia se logra a través de tres tipos principales de repositorios: CRUD Repository, Paging and Sorting Repository y JPA Repository. Estos repositorios no solo reducen la escritura de código repetitivo, sino que también optimizan tiempos de desarrollo y depuración.

**¿Cuáles son los tipos de Spring Repositories y sus funciones?**

CRUD Repository: Este es el repositorio más utilizado por su capacidad para realizar operaciones de creación, lectura, actualización y eliminación (CRUD) sobre cualquier entidad.

Paging and Sorting Repository: Tal como su nombre lo indica, este repositorio permite la paginación y la ordenación de resultados al interactuar con una entidad.

JPA Repository: Extiende los dos repositorios anteriores, incluyendo además operaciones específicas de JPA (Java Persistence API) como flush, lo que facilita tareas más complejas en la base de datos.

**¿Cómo iniciar con los Spring Repositories?**

Para comenzar a usar los Spring Repositories, es necesario seguir unos pasos clave:

Anotar la clase principal, o aquella que tenga la anotación Spring Boot Application, con @EnableJpaRepositories. Esto indica que se utilizarán repositorios de Spring en la aplicación.

Crear una nueva interfaz de repositorio que extienda uno de los repositorios de Spring mencionados. En este caso particular, se utiliza ListCrudRepository para que los métodos de obtención de datos devuelvan listas en lugar de iterables, lo que resulta más intuitivo al trabajar con colecciones de datos.

**¿Cuáles son las ventajas de usar Spring Repositories?**

**Reducción de Código:** Elimina la necesidad de escribir la lógica detallada del acceso a datos, permitiendo centrarse en la lógica de negocios.

**Consistencia y Simplicidad:** Provee una interfaz común y métodos estándar para todas las operaciones básicas sobre los datos.

**Escalabilidad y Flexibilidad:** Facilitan el manejo y la ampliación de la funcionalidad para incluir paginación, ordenación y más, ajustándose a las necesidades del proyecto.

**¿Cómo gestionar las relaciones entre entidades en una base de datos?**

Al crear aplicaciones complejas, frecuentemente las entidades en una base de datos comparten relaciones. Estas pueden ser de uno a uno (1:1), muchos a uno (M:1) o uno a muchos (1:M), entre otras. Es vital entender cómo recuperar dichas relaciones, dado que un mal manejo puede afectar severamente el rendimiento de una aplicación. Un manejo eficiente permite evitar accesos innecesarios o sobrecarga de datos.

**¿Cuándo usar Lazy y Eager en las relaciones?**

Las propiedades de FetchType determinan cómo se recuperan las relaciones:

Lazy: Retrasa la carga de datos hasta que sean específicamente solicitados. Es útil cuando no necesitamos la relación inmediatamente.

Eager: Carga la relación al mismo tiempo que la entidad principal. Usada cuando la relación es vital para el funcionamiento de la operación actual.

**¿Cómo optimizar la recuperación de relaciones?**

De manera predeterminada, JPA y Hibernate gestionan las siguientes configuraciones por tipo de relación:

OneToMany y ManyToMany: Por defecto son Lazy.

ManyToOne y OneToOne: Por defecto son Eager.

Recomendaciones generales:

Limitar la carga de relaciones a solo las necesarias.

Configurar relaciones cómo Lazy siempre que sea posible para mejorar el rendimiento.

Si una relación es ampliamente usada, considerar marcarla como Eager.



**¿Cómo limitar registros con Query Methods en Spring?**

Limitar registros al consultar bases de datos es una tarea fundamental para optimizar la performance de nuestras aplicaciones. En Spring, podemos lograrlo a través de los Query Methods. Estos métodos no solo nos permiten recuperar una cantidad específica de registros, sino también usar elementos de programación funcional como Optional para manejar respuestas.

**¿Cómo asegurarse de que solo un registro se recupere con Query Methods?**

Para limitar la recuperación a un solo registro, no usaremos findAll, sino findFirst. Este método devuelve el primer registro que coincide con las condiciones definidas. En caso de necesitar más de un registro, findTop es la alternativa. Ambos métodos sirven para garantizar el manejo preciso de los datos.

**¿Qué es el Paging and Sorting Repository en Spring Data?**

El Paging and Sorting Repository es una herramienta poderosa dentro de Spring Data que permite gestionar y estructurar grandes volúmenes de datos, haciendo posible paginar y organizar consultas de manera eficiente. Es especialmente útil cuando se trabaja con grandes cantidades de información o simplemente cuando se quiere presentar datos de forma más manejable y accesible.

**¿Qué ventajas ofrece el Paging and Sorting Repository?**

El uso del Paging and Sorting Repository trae consigo varias ventajas significativas:

Eficiencia: Permite manejar y consultar grandes cantidades de datos de manera efectiva, evitando sobrecargar el sistema.

Flexibilidad: Comodidad de configurar el tamaño de página y el número de página según las necesidades del usuario.

Simplicidad: Es fácil de implementar y no requiere código complejo, lo que facilita su integración en aplicaciones existentes.

**¿Cómo implementar un ordenamiento dinámico en un repositorio en Spring?**

Imagina que puedes controlar la manera en que los datos se presentan, no solo la cantidad de información como con la paginación, sino también el orden en que estos aparecen. Eso es precisamente lo que lograremos con el Paging and Sorting Repository en Spring.

**¿Qué beneficios proporciona el uso de Page Unsorting Repository?**

Integrar paginación y ordenamiento dinámico en tu aplicación mejora significativamente la experiencia del usuario final. No solo le permite acceder a grandes volúmenes de datos de manera más efectiva, sino que también le da control sobre la manera en que se presenta la información.

**¿Qué es JPQL y cómo utilizarlo?**

JPQL, o Java Persistent Query Language, es un lenguaje que se utiliza para realizar consultas sobre una base de datos desde las Entities en vez de las tablas tradicionales. Esto permite trabajar de manera más intuitiva para los desarrolladores en Java, ya que se utilizan atributos de objetos en lugar de columnas y tablas.

**¿Cuál es la utilidad de trabajar con queries nativos en Spring Data?**

Trabajar con queries nativos en Spring Data nos proporciona una flexibilidad impresionante, ya que nos permite operar directamente con SQL y aprovechar todas las características que una base de datos particular soporta. A diferencia de los Query Methods o JPQL, que pueden tener limitaciones en algunos escenarios específicos, los queries nativos nos liberan de estas restricciones e incrementan nuestras opciones de implementación.

**¿Qué son las projections y cómo funcionan en Java?**

Las projections son DTOs (Data Transfer Objects) que nos permiten definir una estructura personalizada para recuperar datos específicos de una base de datos. Son especialmente útiles cuando requerimos construir consultas complejas que no se ajustan completamente a los campos de una sola tabla

**¿Cuál es el beneficio de usar projections?**

Las projections ofrecen varias ventajas:

* **Eficiencia**: Se evita cargar entidades enteras con todas sus relaciones, mejorando la performance.
* **Claridad**: Hacen que el código sea más legible y facilitan la comprensión de lo que se está recuperando.
* **Flexibilidad**: Proveen una forma simple de definir y alterar estructuras de datos retornadas sin cambiar la lógica del query.

**¿Qué es un Data Transfer Object (DTO)?**

Un DTO es una clase sencilla cuyo objetivo es transferir datos entre procesos.

**¿Qué es el Spring Expression Language (SPEL)?**

SPEL es un poderoso lenguaje que permite acceder a propiedades de objetos complejos de manera sencilla dentro de consultas @Query. Esto significa que puedes usar un único parámetro, en este caso, el DTO, y acceder a sus propiedades internas mediante expresiones.

Es crucial resaltar la anotación @Transactional, que garantiza que todas las operaciones se realicen dentro de una transacción.

**¿Cómo resolver el error 500 al ejecutar consultas que modifican datos?**

Al ejecutar aplicaciones que modifican datos en la base de datos, puedes encontrarte con el error 500 si no has configurado correctamente las anotaciones. Este error generalmente ocurre si no se usan las anotaciones @Modifying en conjunto con @Query para operaciones de actualización.

**¿Qué es ACID y por qué es vital en las transacciones de bases de datos?**

Cuando trabajamos con bases de datos en nuestras aplicaciones, la integridad y confiabilidad de las transacciones son esenciales para asegurar que la información almacenada esté siempre precisa y segura. ACID es un conjunto de cuatro propiedades que toda transacción debe cumplir para garantizar su confiabilidad.

**Atomicidad**: Esta propiedad asegura que las transacciones sean "todo o nada". Si un error ocurre durante la ejecución, se realiza un rollback para restaurar el estado inicial.

**Consistencia**: Valida que las transacciones solo realicen operaciones que mantienen la integridad de la información. Esto garantiza que las bases de datos no se corrompan con datos inválidos.

**Aislamiento**: Garantiza que las transacciones se ejecuten de manera independiente, evitando que los datos se mezclen entre operaciones concurrentes.

**Durabilidad**: Asegura que la información persista a lo largo del tiempo, incluso si la base de datos se apaga y se vuelve a encender.

**¿Qué es la auditoría en bases de datos y por qué es importante?**

La auditoría en bases de datos es fundamental para el control y el manejo eficiente de los datos. Saber quién realizó cambios, qué cambios se hicieron y cuándo se realizaron es crucial para garantizar la integridad y la seguridad de los datos. En el contexto de las bases de datos, la auditoría ayuda a mantener un registro detallado de todas estas operaciones, lo cual puede ser invaluable en situaciones donde se necesite rastrear acciones o corregir errores.

**¿Qué configuraciones iniciales se necesitan?**

En el main ejecutable:

Incluye @EnableJpaAuditing para habilitar la auditoría.

**¿Cómo auditar una entidad específica?**

Añadir el AuditingEntityListener a la entidad.

**¿Cómo evitar que las fechas de auditoría se incluyan en la respuesta de JSON?**

Las columnas de auditoría pueden no ser relevantes para el usuario final. Utiliza la anotación @JsonIgnore para excluir estas columnas de la respuesta JSON.

**¿Qué beneficios ofrece la auditoría de entidades?**

Con una implementación adecuada de auditoría:

Monitorea y controla cambios en los datos.

Mejora la transparencia en el manejo de información.

Ayuda a proteger contra el acceso no autorizado o modificaciones indeseadas.

Proporciona información valiosa para resolver incidencias futuras.

**¿Qué es un listener personalizado?**

Un listener es una clase con métodos que reaccionan a eventos del ciclo de vida de las entidades.

**¿Cómo utilizar store procedures en Spring Data?**

En el desarrollo de aplicaciones, recurrir a los store procedures es una práctica común para manejar operaciones complejas en bases de datos, garantizando mantenimiento y eficiencia. Spring Data, un proyecto del ecosistema Spring, nos facilita esta tarea a través de la anotación @Procedure, permitiendo declarar y ejecutar store procedures de una manera sencilla.

**¿Qué es un store procedure y cómo lo manejamos?**

Un store procedure es un bloque de código almacenado y reutilizable que permite realizar operaciones complejas en la base de datos, agrupando múltiples instrucciones SQL. Esto puede incluir selecciones, inserciones, o control de transacciones, incrementando tanto la eficiencia como la seguridad de las operaciones.

**¿Qué aspectos debemos verificar antes de ejecutar el Store Procedure?**

Asegúrate de que el store procedure esté correctamente definido en tu base de datos antes de su ejecución.

Utiliza herramientas como Postman para enviar las solicitudes post y verificar su correcto funcionamiento.

Con la anotación @Transactional, gestiona el compromiso de los datos evitando inconsistencias.