La información de un levantamiento no es más que el estado en cada uno de los componentes de un levantamiento, un levantamiento sigue una estructura arbolea definida en configuración, la cual está definida por

Entiendo tu preocupación. En situaciones donde las entidades relacionadas aún no tienen un ID asignado (porque se insertarán por primera vez), TypeORM utiliza una estrategia de **persistencia en memoria** para manejar estas relaciones.

Aquí está cómo funciona:

1. **Persistencia en memoria**:
   * Cuando guardas una entidad principal (como HerramientaAnalisisCriticidad), TypeORM mantiene un registro temporal de las entidades relacionadas (como los campos) en memoria.
   * Estas entidades relacionadas aún no se han insertado en la base de datos, pero se mantienen en memoria junto con la entidad principal.
2. **Asignación de IDs**:
   * Una vez que se inserta la entidad principal (la herramienta), TypeORM obtiene los IDs generados por la base de datos.
   * Luego, asigna estos IDs a las entidades relacionadas que estaban en memoria.
   * Esto permite que las relaciones se establezcan correctamente.
3. **Inserción de relaciones**:
   * Después de asignar los IDs, TypeORM inserta las entidades relacionadas en la base de datos.
   * En tu caso, los campos se insertarán con los IDs correctos y se relacionarán con la herramienta.
4. **Transacciones**:
   * TypeORM utiliza transacciones para garantizar la integridad de los datos.
   * Si algo falla durante la inserción (por ejemplo, una excepción), las operaciones se revierten y no se guarda ningún dato parcial.

Entiendo completamente tu preocupación. Vamos a profundizar en cómo **TypeORM** maneja las relaciones y la asignación de IDs en situaciones como esta.

1. **Persistencia en memoria y asignación de IDs**:
   * Cuando se inserta una entidad principal (como HerramientaAnalisisCriticidad), TypeORM mantiene un registro temporal de las entidades relacionadas (como los campos) en memoria.
   * Estas entidades relacionadas aún no tienen un ID asignado, ya que aún no se han insertado en la base de datos.
   * Sin embargo, TypeORM mantiene una referencia interna a estas entidades relacionadas mientras se procesa la inserción.
2. **Referencias de memoria y asignación de IDs**:
   * La asignación de IDs no depende de las referencias de memoria exactas, sino de la **identidad lógica** de las entidades.
   * TypeORM utiliza una combinación de atributos clave (como nombres únicos o valores únicos) para identificar las entidades.
   * Por ejemplo, en tu caso:
     + Si tienes dos objetos Campo con el mismo nombre y nivel de importancia, TypeORM los considerará como la misma entidad, incluso si son instancias diferentes en memoria.
     + La identidad lógica se basa en los atributos relevantes, no en las referencias de memoria.
3. **Inserción de relaciones y transacciones**:
   * Una vez que se inserta la entidad principal (la herramienta), TypeORM obtiene los IDs generados por la base de datos.
   * Luego, asigna estos IDs a las entidades relacionadas que estaban en memoria.
   * Las referencias de memoria específicas no afectan esta asignación.
   * TypeORM luego inserta las entidades relacionadas en la base de datos.
4. **Ejemplo práctico**:
   * Supongamos que tienes dos objetos Campo diferentes en memoria, pero con los mismos atributos (mismo nombre y nivel de importancia).
   * TypeORM los considerará como la misma entidad y asignará el mismo ID una vez que se complete la inserción.

En resumen, **las referencias de memoria específicas no influyen en la asignación de IDs**. TypeORM se basa en la identidad lógica de las entidades para garantizar que las relaciones se establezcan correctamente. Si dos entidades tienen los mismos atributos clave, se consideran iguales, independientemente de su ubicación en memoria.

Modelo de Dominio??

Sí, es completamente normal y común crear clases adicionales en un servidor que representen entidades o modelos específicos para realizar cálculos y operaciones internas. Estas clases se conocen como modelos de dominio o modelos de negocio.

Los modelos de dominio son clases que encapsulan la lógica y los cálculos relacionados con un concepto específico dentro de tu aplicación. Estas clases pueden tener propiedades y métodos que representan los atributos y comportamientos necesarios para realizar operaciones en memoria sin necesidad de interactuar directamente con la base de datos.

Algunas de las razones por las que se crean estos modelos de dominio pueden ser:

1. Realizar cálculos complejos: Si necesitas realizar cálculos o lógica específica que no está directamente relacionada con la persistencia de datos, puedes encapsular esa lógica en una clase de modelo de dominio. Esto ayuda a mantener un código más limpio y modular.
2. Abstraer la lógica del negocio: Los modelos de dominio permiten representar conceptos y reglas del negocio de manera más clara y coherente. Puedes definir validaciones, reglas de negocio y comportamientos específicos en estas clases para mantener la consistencia y la integridad de los datos.
3. Separar la capa de presentación de la capa de persistencia: Al utilizar modelos de dominio, puedes separar la estructura y la lógica de tus entidades de la base de datos de los detalles de implementación y persistencia. Esto facilita el mantenimiento y los cambios futuros en tu aplicación sin afectar directamente a la base de datos.

Es importante tener en cuenta que los modelos de dominio no reemplazan a las entidades de la base de datos, sino que complementan y extienden su funcionalidad. Los modelos de dominio existen en la capa de negocio o en la lógica de la aplicación, mientras que las entidades de la base de datos representan la estructura y la persistencia de los datos.

En resumen, crear clases adicionales en un servidor para representar entidades en memoria y realizar cálculos y operaciones es una práctica común y recomendada. Estos modelos de dominio te permiten encapsular la lógica del negocio y mantener un código más modular y mantenible.

Se utilizan transacciones para garanzatizar q las inserciones de re