Guía de Mapeo de Herencia en Hibernate con Anotaciones

Hibernate permite mapear relaciones de herencia en Java a tablas de base de datos. Existen distintas estrategias que determinan cómo se almacenarán las clases hijas en relación con la clase padre. La elección de la estrategia depende del rendimiento, la normalización y la complejidad deseada en el esquema.

1. Estrategia SINGLE_TABLE

Con esta estrategia, todas las clases de la jerarquía se almacenan en una sola tabla. Se diferencia el tipo de entidad mediante una columna discriminadora.
Ejemplo:

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "tipo", discriminatorType = DiscriminatorType.STRING)
public abstract class Empleado { ... }

@Entity
@DiscriminatorValue("ADMIN")
public class Administrador extends Empleado { ... }

@Entity
@DiscriminatorValue("DEV")
public class Desarrollador extends Empleado { ... }
```

2. Estrategia TABLE_PER_CLASS

Cada clase hija tiene su propia tabla independiente, incluyendo las columnas heredadas del padre. No existe una tabla común para el padre.
Ejemplo:

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Empleado { ... }

@Entity
public class Administrador extends Empleado { ... }

@Entity
public class Desarrollador extends Empleado { ... }
```

3. Estrategia JOINED

Cada clase de la jerarquía tiene su propia tabla. La tabla hija almacena solo sus atributos específicos y se une con la tabla padre mediante una clave foránea. Es la estrategia más

normalizada y flexible, aunque puede requerir más joins en las consultas.
Ejemplo:

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Empleado { ... }

@Entity
public class Administrador extends Empleado { ... }

@Entity
public class Desarrollador extends Empleado { ... }
```

Comparación de Estrategias

1. SINGLE_TABLE: Mayor rendimiento, menos joins, pero la tabla puede crecer mucho y tener columnas nulas.
2. TABLE_PER_CLASS: Consultas rápidas por clase, pero duplicación de columnas y sin tabla unificada.
3. JOINED: Normalización adecuada y sin duplicación, pero con más joins en consultas.

La estrategia a elegir depende del contexto: si se prioriza el rendimiento con una tabla sencilla, SINGLE_TABLE es adecuada. Para consultas específicas por tipo con independencia, TABLE_PER_CLASS es útil. Si se busca normalización y consistencia, JOINED es la mejor opción.