Acceso a datos

ORM – Mapeo objeto-relacional

Persistencia de objetos – Desfase objeto/relacional – Herramientas ORM

IES Clara del Rey – Madrid

Persistencia de objetos

La persistencia es la capacidad de que los datos manejados por un programa sobrevivan a la aplicación. Que, aunque el programa finalice, los datos se mantengan para ser utilizados la próxima vez que se ejecute la aplicación.

Los lenguajes de programación modernos son, prácticamente todos, orientados a objetos, y maneja los datos utilizando interfaces, clases y objetos.

Lo más habitual es almacenar la información en bases de datos.

Hoy día, pese a la creciente popularidad de bases de datos NoSql, la mayoría de las bases de datos son relacionales, basadas en tablas y filas. Para almacenar objetos en un modelo relacional hay que hacer ciertas conversiones y adaptaciones.

Desfase objeto-relacional

La expresión desfase o impedancia objeto-relacional, hace referencia a los problemas que hay que resolver para guardar objetos (o un grafo de objetos conectados entre sí) en una base de datos relacional.

Los principales problemas son:

- Granularidad – Número de tablas y columnas vs clases y atributos.
- Herencia – En los SGBD no existe la herencia.
- Identidad e igualdad – Claves primarias vs igualdad de objetos.
- Asociación y navegación de relaciones – Claves ajenas vs relaciones (agregación y composición) de objetos
- Diferencias en tipos de datos – Tipos SQL vs tipos Java (u otro lenguaje)

Desfase objeto-relacional – Granularidad

Hace referencia a la diferencia en el número de clases en un modelo de objetos vs el número de tablas que hay en la base de datos.

Podemos tener más clases que tablas. Por ejemplo, una clase "Cliente", otra "Direccion", y otra "Provincia", con sólo una tabla "Clientes" en la base de datos.

También podría ser al revés. Podríamos tener una única clase "Cliente" que obtuviera los datos de varias tablas, por ejemplo, una tabla "Clientes" y otra "DetallesClientes".

Desfase objeto-relacional – Herencia

El concepto de herencia no existe en las bases de datos relacionales.

Y relacionado con la herencia, los conceptos de las interfaces o las clases abstractas tampoco existen.

El sistema más habitual para trasladar la herencia a una base de datos relacional es el de columnas discriminantes.

Digamos que tenemos la clase abstracta "Animal", de la que heredan la clase "Perro" y la clase "Gato". En la base de datos podríamos tener una tabla "Animales", con una columna "TipoAnimal", en la que se identifique si la fila se refiere a un gato o a un perro.

Adicionalmente, se puede tener tablas adicionales para los atributos específicos de cada subclase.

Desfase objeto-relacional
Identidad e igualdad

En las bases de datos relacionales, en términos generales, se considera que dos filas son iguales si tienen la misma clave primaria. Incluso cuando la clave primaria es una clave surrogada o sustituta, su unicidad coincide con el índice único de una clave natural.

En POO, sin embargo, gestionamos dos conceptos diferentes, que permiten comparar objetos de dos formas distintas:

- Identidad. Referencias a objetos. Cuando comparamos referencias (objetoA == objetoB) estamos comparando las referencias, mirando si son el mismo objeto.
- Igualdad. Basado en el valor o estado del objeto. Cuando comparamos objetos (objetoA.equals(objetoB)), comparamos en función del contenido o atributos del objeto.

Desfase objeto-relacional
Asociación y navegación de relaciones

La asociación representa la conexión entre clases de nuestro modelo orientado a objetos. Se consigue usando referencias de un objeto a otros objetos.

Esto en BD relacionales se consigue con las relaciones (claves ajenas).

La navegación permite movernos por estas referencias / relaciones, pudiendo obtener el objeto u objetos relacionados con otro.

El problema de la navegación es que cuando tenemos una relación n a n entre dos tablas "ObjetosA" y "ObjetosB" en una base de datos, no hay una forma de representar esta relación en clases. Se tiene que dividir en dos asociaciones entre las clases, una en cada sentido.

Desfase objeto-relacional
Diferencias entre tipos de datos

Puede que no haya una equivalencia directa entre el tipo de dato de un atributo de una clase y el tipo de dato de una columna SQL.

Por ejemplo, en la mayoría de los lenguajes orientados a objeto, las cadenas o "strings" no tienen límite de longitud. Sin embargo, en SQL sí pueden tener límite, y normalmente hay un límite de longitud en las columnas de tipo varchar.

Otro caso similar se da con boolean, que en algunos gestores de BBDD relacionales ni si quiera existe. Puede existir un tipo alternativo, como "bit" en MS SQL Server o el char en Oracle, que no tuvo boolean hasta la versión 23c (2023).

Persistencia de objetos de forma "manual"

Se trata de desarrollar una serie de clases que son las responsables de:

- Leer datos de la base de datos y materializar (crear) los objetos necesarios para representar los datos leídos.
- Escribir en la base de datos el contenido (atributos) de los objetos, de forma que puedan ser leídos y materializados de nuevo.

El enfoque habitual es una serie de clases de acceso a datos, una por cada clase del programa orientado a objetos, que se encargan de todos los aspectos relacionados con la persistencia de esa clase.

¿Problemas? Requiere mucho código que hay que desarrollar y que, aunque no es complicado, si requiere muchas pruebas.

ORM – Object / relational mapping

El mapeo objeto – relacional es un mecanismo que permite mapear la estructura de información de una base de datos relacional a una estructura lógica de clases, y viceversa.

El ORM crea, de forma más o menos automática, estructuras, como colecciones de objetos, clases o interfaces que permiten el acceso a la base de datos.

Los ORM liberan al programador de escribir gran cantidad de código para acceso a la base de datos: conexiones (Connection), consultas SQL, DML o DDL (PreparedStatement), objetos para acceso a datos (ResultSet), etc.

En función del ORM puede que sea necesario usar un lenguaje similar al SQL para realizar consultas.

Ventajas y desventajas de usar un ORM

Ventajas

- Menor tiempo de desarrollo, lo que implica menor coste.
- Abstracción (independencia) de la base de datos utilizada.
- No hay que saber SQL para la mayoría de las operaciones.

Desventajas

- Se pierde el detalle de lo que está pasando al acceder a la BD.
- Pueden ser más lentos, sobre todo en consultas complejas.

En general, los beneficios superan a los inconvenientes. Algunos de los inconvenientes se pueden minimizar si utilizamos SQL nativo para las consultas problemáticas, algo que la mayoría de los ORM permiten.

Estrategias de mapeo
Code first vs Database first

A la hora de usar un ORM suele haber dos formas de hacerlo:

- Code first
  - El modelo (clases con sus anotaciones) define la estructura de la BD.
  - El ORM crea / modifica / actualiza tablas en la BD para que se ajusten a la estructura de clases.
- Database first
  - Se usa el ORM para conectar a una BD pre-existente.
  - El modelo (clases con sus anotaciones) debe adaptarse a la BD.
  - El ORM puede verificar que los mapeos son correctos, pero no cambia la BD.