Mapeo Objeto Relacional

Marko Antonio Rivas Rios

Universidad Privada de Tacna
marko.rivas98@gmail.com

Abril 12, 2019

Abstract

La asignación de objetos relacionales es una técnica que le permite asignar cada una de las filas de nuestras tablas en la base de datos de objetos, donde las columnas de la tabla corresponden a las propiedades de estos objetos. La técnica, utilizada en la programación para convertir los tipos de datos con los que trabajan en un lenguaje orientado a objetos, los tipos de datos con los que trabajan en un sistema de base de datos relacional para la persistencia de datos en el objeto de mapeo-relacional.

El ORM también se conoce con el nombre de ORM a los marcos que la implementación, como EntityFramework en .NET, Hibernate en Java y ActiveRecord en Ruby. Todos estos son nombres de ORM utilizados, archivos de configuración en XML o anotaciones dentro del código para configurar la relación que existe entre una clase de nuestro lenguaje orientado a objetos y una tabla en la base de datos.

El uso de una herramienta introduce una capacidad de abstracción entre la base de datos y el desarrollador, el ORM evita que el desarrollador tenga que escribir las consultas "a mano" para recuperar, filtrar, agrupar, eliminar, actualizar o insertar datos en la base de datos. La tarea de ORM será la de transformar las operaciones realizadas en el nivel orientado a objetos en sentencias SQL con la base de datos que pueda funcionar.

I. Introducción

ORM es el mapeo objeto-relacional (más conocido por su nombre en inglés, Object- Relational mapping), consiste en una técnica de programación para convertir datos entre el lenguaje de programación orientado a objetos utilizado y el sistema de base de datos relacional utilizado en el desarrollo de aplicaciones. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

Entre estos paquetes comerciales tenemos una lista alfabética de los principales motores de mapeo objeto relacional, tales como: ColdFusion, Common Lisp, Java, JavaScript, .NET, Perl, PHP, Python, Ruby, Smalltalk, C++.

El problema que surge, porque hoy en día prácticamente todas las aplicaciones están diseñadas para usar la Programación Orientación a Objetos (POO), mientras que las bases de datos más extendidas son del tipo relacional y estas solo permiten guardar tipos de datos primitivos (enteros, cadenas de texto) por lo que no se puede guardar de forma directa los objetos de la aplicación en las tablas, sino que estos se deben de convertir antes en registros, que por lo general afectan a varias tablas. En el momento de volver a recuperar los datos, hay que hacer el proceso contrario, se deben convertir los registros en objetos.

En este punto es que se muestra la importancia del ORM, ya que este se encarga de forma automática, de convertir los objetos en registros y viceversa, simulando así tener una base de datos orientada a objetos.

II. Objetivos

El desarrollo de software busca mejorar la productividad de las empresas por medio de la automatización y el uso de herramientas, las empresas que se dedican al desarrollo de software tienen como objetivo ayudar a otras empresas en la automatización y desarrollo de herramientas para este fin, pero muchas veces las mismas olvidan su propia productividad. Las Herramientas ORM (Object Relational Mapping) se han ideado con este fin, al evitar repetir muchas líneas de programación en la capa de abstracción, pero es necesario considerar que cada una tiene su estándar o su propio lenguaje por lo cual se requeriría tiempo adicional para ocuparlas.

El desarrollo de este ORM tiene como objetivo proveer una herramienta acorde a las necesidades y el estándar de programación de la empresa Interfaces, logrando que, durante el desarrollo de software, este se centre en otros aspectos más relevantes, como el diseño de la base de datos, interfaz de usuario y la capa de negocios, olvidándonos casi por completo de la capa de abstracción. Con el uso de la herramienta ORM se logró reducir el tiempo de desarrollo de la aplicación de prueba.

Realizar el modelo y modo de implementación del ORM en un java-object.

Implementar la recuperación de datos automáticamente de los java-objects.

Implementar la conectividad y las consultas autogeneradas a la Base de Datos a partir de los modelos heredados del modelo base del ORM.

Implementar la conversión automática de los resultados a los modelos java-object.

Realizar pruebas de validación al ORM.

III. Marco Teórico

i. ¿Qué es un ORM?

Hoy hablamos del Mapeo Objeto-Relacional o como se conocen comúnmente, ORM (del inglés Object Relational Mapping). Algunos de vosotros ya sabréis que son pero, para aquellos que no los conozcan, un ORM te permite convertir los datos de tus objectos en un formato correcto para poder guardar la información en una base de datos (mapeo) creándose una base de datos virtual donde los datos que se encuentran en nuestra aplicación, quedan vinculados a la base de datos (persistencia).

Si alguna vez has programado alguna aplicación que se conecta a una base de datos, habrás podido comprobar lo laborioso que es transformar toda la información que recibes de la base datos, principalmente en tablas, en los objetos de tu aplicación y viceversa. A ésto se le denomina mapeo. Utilizando un ORM este mapeo será automático, es más, será independiente de la base de datos que estés utilizando en ese momento pudiendo cambiar de motor de base de datos según tus necesidades.Veamos un ejemplo. Supongamos que tenemos una tabla de clientes. En nuestra aplicación queremos hacer las funciones básicas sobre base de datos CRUD (del inglés Create, Read, Update and Delete) Crear, Obtener, Actualizar y Borrar. Cada operación corresponde con una sentencia SQL.

- Crear: INSERT
- Obtener: SELECT
- Actualizar: UPDATE
- Borrar: DELETE

ii. Herramientas ORM

En el mercado podemos encontrar diversas herramientas tanto de pago como de uso libre. Algunos programadores, prefieren invertir tiempo en desarrollar su propia herramienta ORM usando patrones de diseño bien conocidos como son el Repository o el Active Record.

El patrón Repository se soporta sobre la definición de un repositorio para separar la lógica que recupera los datos de la base de datos de la lógica de negocio basada en objetos. Este repositorio hace de puente entre los datos y las operaciones basadas en objetos, eliminando dependencias tecnológicas y facilitando el acceso a datos de cualquier tipo.

Por otro lado, está el patrón Active Record. Es un patrón en el cual, el objeto contiene los datos que representan a un registro de nuestra tabla o vista relacional, además de encapsular la lógica necesaria para acceder a la base de datos. De esta forma el acceso a datos se presenta de manera uniforme a través de la aplicación (lógica de negocio + acceso a datos en una misma clase).

Las herramientas ORM que hay en el mercado son diversas. Algunas están ligadas al lenguaje de programación orientado a objetos específico.

Algunos ejemplos específicos serían:

• Para Java: Hibérnate, iBatis, Ebean, Torque
• Para .Net: nHibernate, Entity Framework, DataObjects.NET
• Para PHP: Doctrine, Propel , Torpor
• Para Python: SQLObject, Django, Tryton

iii. Ventajas e Inconvenientes de las Herramientas ORM

Las herramientas ORM ofrecen ventajas para el programador como son:

• Rapidez en el desarrollo
• Abstracción de la base de datos utilizada

• Seguridad de la capa de acceso
• Facilidad para el mantenimiento del código
• Lenguaje propio para la realización de consultas

Pero no todos son ventajas, también tienen una serie de inconvenientes que deben valorarse, como son:

• El aprendizaje del lenguaje de la herramienta ORM puede resultar ser complejo ya que, para poder sacar el máximo partido a la herramienta, es necesario conocer en profundidad cómo funciona la misma.
• En entornos de gran carga, este tipo de solución penaliza el rendimiento debido a los procesos de transformación de las consultas que se hagan hacia la base de datos.

iv. Herramienta Entity Framework

Entity Framework es el ORM (Object-Relational Mapper) de Microsoft, con versiones tanto para la plataforma .NET "tradicional" como para .NET Core.
Como vimos en el artículo del enlace anterior, en el que se explicaba con detalle qué es un ORM, este tipo de software puede funcionar de varias maneras diferentes a la hora de "mapear" las clases de nuestro programa orientado a objetos y las tablas en la base de datos.
Entity Framework no es una excepción, y nos ofrece diversas maneras de trabajar con los datos desde nuestros programas. Cada una tiene un enfoque diferente y es interesante para ciertos casos concretos, además de tener sus beneficios y problemas.
Vamos a dar un repaso rápido a los 3 modos de trabajo principales de Entity Frame-

work para ver en qué consisten y sus ventajas e inconvenientes.
Es importante tener en cuenta que las capacidades de Entity Framework en .NET "tradicional" (EF6) y en .NET Core (EF Core) son completamente diferentes. Así, los tres modos de trabajo descritos a continuación están completamente soportados en EF6, pero EF Core solamente soporta "Code First" y muy poquito de "Database First". "Model First" en .NET Core ni está ni se le espera. A continuación lo detallaremos más.

IV. Análisis

Comencemos analizando JDBC(API de Java)como solución. El mayor beneficio es el rendimiento dado que programado adecuadamente, con store procedures y al no tener capas intermedias que pasar, el rendimiento aumenta a sacrificio de muchas características que son de suma importancia en una arquitectura empresarial. JDBC es una API de bajo nivel y sus prestaciones son para funciones de bajo nivel. Cuando el modelo de datos es simple la solución más fácil es JDBC.

Los frameworks ORM ya están consolidados, probados en el mercado y existen muchas herramientas para desarrollar con estos frameworks. La mayoría de ellos permiten la definición de las entidades generalmente a través de ficheros XML o anotaciones. A partir de esa definición los ORM son capaces de extraer la definición de esquema de la base de datos necesaria para representar ese modelo de objetos.

Los ORM permiten el mapeo de estos esquemas de base de datos a objetos de modo que a partir de las tablas de base de datos se pueden generar las clases Java, necesarias para modelar dicho esquema con todas las relaciones de jerarquía que estén presentes en el mismo.
Obviamente esto supone una ventaja grandísima ya que la cantidad de código necesaria para realizar todas esas funciones es realmente considerable.

V. Conclusiones

El uso de un ORM es una alternativa sumamente efectiva a la hora de trasladar el modelo conceptual (orientado a objetos) al esquema relacional nativo de las bases de datos SQL. Evita la inclusión de sentencias SQL embebidas en el código de la aplicación, lo que a su vez facilita la migración hacia otro sistema gestor de bases de datos. Incorpora una capa de abstracción entre el modelo relacional físico y la capa de negocios de la aplicación. Al ser realizado, en esta capa, de manera automática la conversión de instrucciones orientadas a objetos, a sentencias SQL, minimiza la ocurrencia de errores humanos.

De cualquier modo, utilizar un ORM no debe ser considerado una panacea, sino que debe usarse a discreción; teniendo en cuenta las particularidades de cada problema a modelar. En determinados casos no es recomendable el uso de un ORM, sobre todo cuando se imponen tiempos de respuesta mínimos o se requiere una menor sobrecarga. En estos casos lo más conveniente es el uso de un microORM; evitando siempre que sea posible las inyecciones de SQL Inline.

Lo anteriormente expuesto libera a los desarrolladores de aplicaciones de la responsabilidad de conocer las múltiples variantes de SQL que existen en función del gestor de bases de datos que se utilice. No obstante, en escenarios en que se necesite hacer un uso más eficiente del sistema de almacenamiento de información, personalizado de acuerdo a las necesidades de la aplicación, y se escoja como gestor de bases de datos una variante NoSQL no es necesaria la utilización de un ORM.

References

Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330.