OR/M (Object Relational Mapping)

Robles Flores, Anthony Richard (2016056192), Porlles Carrillo, Diego Armando (2015050948), Sandoval Blas, Jesus Enrique (2016054467), Quispe Mamani, José Luis (2015053235)

Tacna, Perú

Abstract

Object-Relational Mapping is a technique used in programming to convert the types of data with which a language oriented to work objects to data types with which a relational database management system works for data persistence. Basically consists The Object-Relational Mapping is considered to be a technique that is used in programming in order to convert the types of data with which an object-oriented language works to data types with which a database management system works of the relational type. It consists of a technique that will allow us to map each of the rows of our tables in the database to objects, where the columns of the table correspond to the properties of these objects.

1. Resumen

El Mapeo Objeto-Relacional se le considera que es una técnica que se usa en programación con la finalidad de convertir los tipos de datos con los que trabaja un lenguaje orientado a objetos a tipos de datos con los que trabaja un sistema gestor de base de datos del tipo relacional. Consiste en una técnica que nos permitirá mapear cada una de las filas de nuestras tablas en la base de datos a objetos, en donde las columnas de la tabla corresponden a las propiedades de estos objetos.

2. Introducción

En este articulo explicaremos comenzando sobre las definiciones existentes de Mapeo de Objeto Relacional y como esto influye en la parte de la

programacion a la confersion de datos para el uso en un sistema gestor de base de datos , hablando a su ves de Patron Repository , las ventajas de usar esta tecnica como tambien las desventajas de usarla , dando a su vez nuestro analisis de acuerdo a las fuentes relacionadas con sus respectivas conclusiones de lo entendido.

3. Marco Teórico

3.1. Definición

ORM es el nombre de la técnica, también se suele conocer como ORMs a los frameworks que la implementan, como EntityFramework en .NET, Hibernate en Java y ActiveRecord en Ruby. Estos ORMs usan convenciones de nombres, 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 de ORM introduce una capa de abstracción entre la base de datos y el desarrollador, ya que gracias a este mapeo, los ORM evitan que el desarrollador tenga que escribir consultas .ª mano"para recuperar, filtrar, agrupar, eliminar, actualizar o insertar datos en la base, será tarea del ORM transformar las operaciones hechas a nivel orientado a objetos a sentencias SQL con las que la base de datos puede trabajar.[1]
Entre las ventajas que ofrecen los ORM se encuentran: rapidez en el desarrollo, abstracción de la base de datos, reutilización, seguridad, mantenimiento del código, lenguaje propio para realizar las consultas. No obstante los ORM traen consigo algunas desventajas como el tiempo invertido en el aprendizaje. Este tipo de herramientas suelen ser complejas por lo que su correcta utilización requiere un espacio de tiempo a emplear en conocer su funcionamiento adecuado para posteriormente aprovechar todo el partido que se le puede sacar. Otra desventaja es que las aplicaciones suelen ser algo más lentas. Esto es debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos. En el mapeo objeto-relacional encontramos el uso de algunos patrones de diseño como el Repository y el Active Record.[2]

3.2. Patrón Repository

El patrón Repository utiliza un repositorio para separar la lógica que recupera los datos y los mapea al modelo de entidades, de la lógica del negocio

que actúa en el modelo. El repositorio media entre la capa de fuente de datos y la capa de negocios de la aplicación; encuesta a la fuente de datos, mapea los datos obtenidos de la fuente de datos a la entidad de negocio y persisten los cambios de la entidad de negocio a la fuente de datos. En la figura 1 se muestra un esquema patrón Repository.

Los repositorios son puentes entre los datos y las operaciones que se encuentran en distintos dominios. Un repositorio elabora las consultas correctas a la fuente de datos y mapea los resultados a las entidades de negocio expuestas externamente. Los repositorios eliminan las dependencias a tecnologías específicas proveyendo acceso a datos de cualquier tipo.

El patrón de diseño Repository puede ayudar a separar las capas de una aplicación web ASP.NET ya que provee una arquitectura de 3 capas separadas, lo que mejora el mantenimiento de la aplicación y ayuda a reducir errores. Además facilita las pruebas unitarias.[4]

3.3. Patrón Active Record

Active Record es un patrón en el cual, el objeto contiene los datos que representan a una fila (o tupla) de nuestra tabla o vista, 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). En la figura 2 se muestra un esquema del patrón Active Record.

Una clase Active Record consiste en el conjunto de propiedades que representa las columnas de la tabla más los típicos métodos de acceso como las operaciones CRUD (Create, Read, Update, Delete), búsqueda (Find), validaciones, y métodos de negocio.

Este patrón constituye la aproximación más obvia, poniendo el acceso a la base datos en el propio objeto de negocio. De este modo es evidente como manipular la persistencia a través de él mismo. Gran parte de este patrón viene de un Domain Model y esto significa que las clases están muy cercanas a la representación en la base de datos. Cada Active Record es responsable de sí mismo, tanto en lo relacionado con persistencia como en su lógica de negocio.

[3]

3.4. SubSonic y NHibernate

Las herramientas ORM (Object-Relational Mapping) se utilizan para convertir los datos entre el sistema de tipos utilizado en un lenguaje de progra-

mación orientado a objetos y el utilizado en una base de datos relacional, a través de un motor de persistencia. Esto se traduce en que las herramientas ORM permiten almacenar objetos en una base de datos de forma permanente, y extraerlos cuando sea necesario.

Subsonic es una herramienta ORM libre, de código abierto creada por Rob Conery. Inspirado por Ruby on Rails. Subsonic toma un acercamiento minimalista al código y enfatiza convención sobre configuración. Aunque se inspira en Ruby on Rails, no es una parte de él, sino que toma las ideas de Ruby on Rails y las adapta a la plataforma ya existente ASP.NET.

A diferencia de otros ORM, SubSonic usa el método de la generación y compilación del nivel de acceso de datos en lugar de realizar una asignación basada en reflexión en el tiempo de ejecución. Al igual que muchos ORM, SubSonic soporta los sistemas de bases de datos más comunes, entre ellos: SQL Server, MySQL, SQLite, Oracle, IBM DB2, PostgreSQL, SQL CE, VistaDB y también hace muy fácil la migración del acceso a datos entre estos.

NHibernate es la conversión de Hibernate de lenguaje Java a Csharp para su integración en la plataforma .NET. Al igual que Hibernate, NHibernate es una herramienta de ORM que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML). Hibernate fue una iniciativa de un grupo de desarrolladores dispersos alrededor del mundo conducido por Gaving King.

Al usar NHibernate para el acceso a datos el desarrollador se asegura de que su aplicación es agnóstica en cuanto al motor de base de datos a utilizar en producción, pues NHibernate soporta los más habituales en el mercado: MySQL, PostgreSQL, Oracle, MS SQL Server, etc. Sólo se necesita cambiar una línea en el fichero de configuración para que podamos utilizar una base de datos distinta. Ambas herramientas tienen sus ventajas y desventajas, pero generalmente la selección de uno u otro es una cuestión de preferencia personal.[4]

3.5. Ventajas

- Rapidez en el desarrollo
- Abstraccion de la base de datos
- Mejora Continua: la cultura DevOps se centra en la mejora continua

para minimizar el desperdicio. Continuamente acelera la mejora del producto o los servicios ofrecidos.

- Reutilización.
- Seguridad.
- Mantenimiento del código.
- Lenguaje propio para realizar las consultas.

[5]

3.6. Desventajas

- Tiempo utilizado en el aprendizaje. Este tipo de herramientas suelen ser complejas por lo que su correcta utilización lleva un tiempo que hay que emplear en ver el funcionamiento correcto y ver todo el partido que se le puede sacar.
- Aplicaciones algo más lentas. Esto es debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá de transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.[6]

4. Conclusiones

El uso de este modelo es muy efectivo ya que no evita la inclusion de sentencias SQL embebidas en el codigo de la aplicacion, lo que facilita que la migracion a otro sistema gestor de base de datos. Lo cual tambien es una ayuda ya que evita que se cometan muchos errores humanos que comunmente suelen ocurrir.

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.

En resumen, en dependencia del gestor de bases de datos a emplear, se recomienda siempre que sea posible y sea factible la utilización de un ORM.

Referencias

[1] 2008, V. S. (ne). The microsoft developer network. Recuperado de http://cort.as/-QSJ3. Accedido 18-08-2019.

[2] Guardado, I. (2010). Introducción a object-relational mapping. Recuperado de http://cort.as/-QSHx. Accedido 18-08-2019.

[3] IT, G. (ne). Patrones de acceso a datos: Active record 2009. Recuperado de http://grimpidev.wordpress.com/2008/11/22/patrones-de-acceso-a-datos-active-record/ . Accedido 15-09-2019.

[4] Local Help, V. S. (ne). Msdn, the microsoft developer network 2008. Recuperado de http://msdn.microsoft.com/en-us/ . Accedido 15-09-2019.

[5] Pariseau, B. (2017). Ventajas omr. Recuperado de https://bit.ly/2jVHzk9. Accedido 30-08-2019.

[6] Technologies, C. (2014). Desventajas orm. Recuperado de https://bit.ly/2lP06Px. Accedido 30-08-2019.