Análisis e implicaciones de la implementación del Mapeo Relacional de Objetos en la Programación Orientada a Objetos

Miguel S. Gómez-Díaz[1], Francisco J. Casillas-Rodríguez[1], Laura Juárez Guerra[2], Elizabeth Castellanos Nolasco[2], Ubaldo Uribe-López[2]

[1]Universidad de Guadalajara, Centro Universitario de los Lagos, Lagos de Moreno, Jalisco, México.
[2]Tecnológico Nacional de México campus León, León, Guanajuato, México.

Resumen

Uno de los desafíos de usar lenguajes y bases de datos de Programación Orientada a Objetos (POO) es la complejidad de alinear el código de programación con las estructuras de la base de datos. Este trabajo tiene como objetivo, mostrar las principales características de la implementación del Mapeo Relacional de Objetos como herramienta para la simplificación del uso de la Programación Orientada a Objetos y el almacenamiento de registros en Bases de Datos.

Abstract

One of the challenges of using Oriented Object Programming (OOP) languages and databases is the complexity of aligning programming code with database structures. The objective of this work is to show the main characteristics of the implementation of Object Relational Mapping as a tool for simplifying the use of Oriented Object Programming and the storage of records in Databases.

Palabras clave: Mapeo Relacional de Objetos, Programación Orientada a Objetos, SQL, Modelo Relacional, CRUD.
Keywords: Object Relational Model, Oriented Object Programming, SQL, Relational Model, CRUD.

1. INTRODUCCIÓN

El Mapeo Relacional de Objetos (MRO) es una técnica que crea una capa entre el lenguaje y la base de datos, lo que ayuda a los programadores a trabajar con datos sin el paradigma POO. El desafío para los desarrolladores de Programación Orientada a Objetos es comprender y programar el lenguaje de consulta estructurado (SQL) para conectar su aplicación a una base de datos SQL. Los programadores familiarizados con SQL pueden escribir código de acceso a datos. Esta codificación SQL sin procesar puede llevar mucho tiempo, ya que el programador debe extraer datos de las líneas de código. Los generadores de consultas SQL agregan una capa de abstracción al código SQL para proporcionar más información sobre los datos.

2. MARCO TEÓRICO

Programación Orientada a Objetos

La Programación Orientada a Objetos es un modelo de programación que utiliza objetos, que interactúan mediante mensajes para la solución de un problema. Se basa en el concepto de crear un modelo del problema objetivo en sus programas. La Programación Orientada a Objetos reduce los errores y fomenta la reutilización del código [1]. Hay muchos lenguajes orientados a objetos. Los objetos allí definidos tienen las siguientes propiedades:

1. **Identidad.** Cada objeto debe ser distinguido y ello debe poder demostrarse mediante pruebas.

2. **Estado.** Cada objeto debe ser capaz de almacenar el estado. Para este fin, existen atributos, tales como variables de instancias y campos.
3. **Comportamiento.** Cada objeto debe ser capaz de manipular su estado. Para este fin existen métodos.

La mayoría de los lenguajes incluyen las características siguientes para dar soporte a la Programación Orientada a Objetos:

4. **Creación de objetos basada en clases.** Las clases son marcos para la creación de objetos. Los objetos son estructuras de datos con el comportamiento asociado.
5. **Herencia con polimorfismo.** Dan soporte a la herencia individual y múltiple. Todos los métodos de instancias pueden ser polimórficos y se pueden alterar temporalmente mediante subclases.
6. **Encapsulación de datos.** Esta permite ocultar los atributos. Cuando se ocultan los atributos, se puede acceder a los mismos desde fuera de la clase únicamente mediante los métodos de la clase. Las clases implementan métodos para modificar los datos. Esto permite que no sea tan fácil poder alterar los valores de los atributos.

Base de datos relacional

Una base de datos relacional es un tipo de base de datos que almacena y proporciona acceso a puntos de datos relacionados entre sí. Las bases de datos relacionales se basan en el modelo relacional, una forma intuitiva y directa de representar datos en tablas [2]. En una base de datos relacional, cada fila en una tabla es un registro con una ID única, llamada clave. Hay cuatro características clave que definen las transacciones de bases de datos: atomicidad, uniformidad, aislamiento y persistencia. Estos son comúnmente conocidos como ACID, las siglas en inglés.

a. La atomicidad define todos los elementos que conforman una transacción completa de base de datos.
b. La uniformidad o también conocida como consistencia, define las reglas para mantener los puntos de datos en un estado correcto después de una transacción.
c. El aislamiento impide que el efecto de una transacción sea visible a otros hasta que se establezca el compromiso, a fin de evitar confusiones.
d. La durabilidad garantiza que los cambios en los datos se vuelvan permanentes cuando la transacción se haya fijado y hayamos llegado a un compromiso.

Modelo Relacional

El modelo relacional significa que las estructuras de datos lógicas (las tablas de datos, las vistas y los índices) están separadas de las estructuras de almacenamiento físicas. La diferencia entre el nivel lógico y físico también se aplica a las operaciones de bases de datos, que son acciones bien definidas que permiten que las aplicaciones manipulen datos y estructuras de bases de datos [2,3]. Las operaciones lógicas permiten que una aplicación especifique el contenido que necesita, mientras que las operaciones físicas determinan cómo se accede a esos datos y cómo se realizan las tareas. Para garantizar la precisión y el acceso continuo a los datos, las bases de datos relacionales siguen ciertas reglas de integridad. Por ejemplo, una restricción puede especificar que una tabla no debe permitir filas duplicadas para evitar que se ingrese información incorrecta en la base de datos. La figura 1 muestra un ejemplo de un modelo de base de datos relacional y sus conexiones asociadas por tabla.

Concepto CRUD de Bases de Datos

El concepto CRUD está estrechamente vinculado a la gestión o manipulación de datos digitales. CRUD hace referencia a un acrónimo en el que se reúnen las primeras letras de las cuatro operaciones fundamentales de aplicaciones persistentes en sistemas de bases de datos:

- Create (Crear registros)
- Read (Leer registros)
- Update (Actualizar registros)
- Delete (Borrar registros)

Para los expertos, las operaciones son las herramientas de acceso típicas e indispensables para comprobar, por ejemplo, los problemas de la base de datos, mientras que, para los usuarios, CRUD significa crear una cuenta (create) y utilizarla (read), actualizarla (update) o borrarla (delete) en cualquier momento.

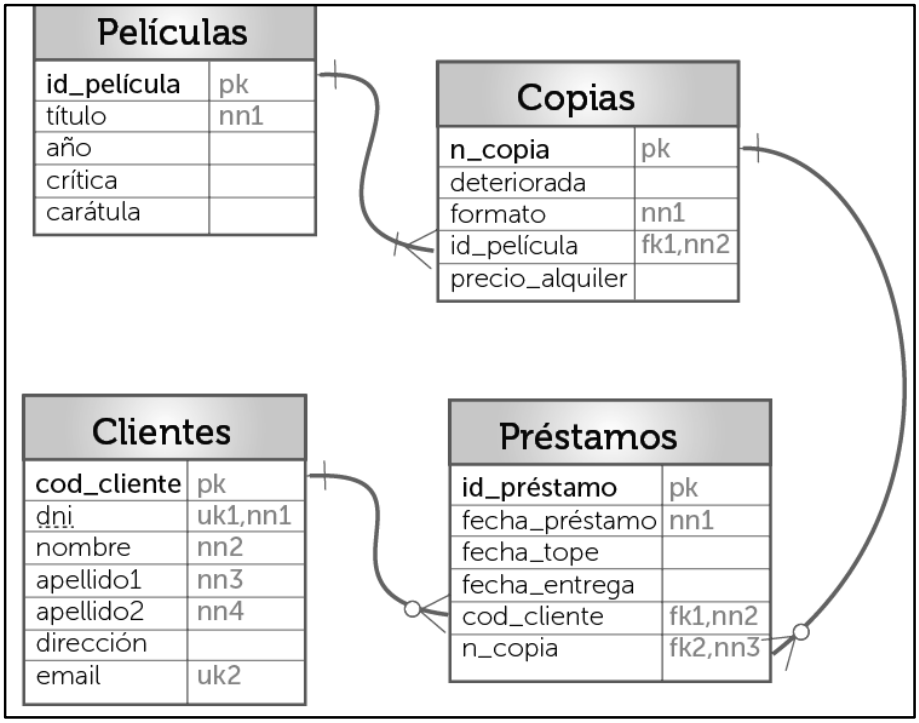

Figura 1. Diagrama representativo de los actores del Modelo Relacional.

Mapeo Relacional de Objetos

El Mapeo Relacional Objeto (MRO) o Object Relational Mapping con su acrónimo en inglés (ORM) es un modelo de programación que permite mapear las estructuras de una base de datos relacionales (SQL Server, Oracle, MySQL, etc.) sobre una estructura lógica de entidades con el objetivo de simplificar y acelerar el desarrollo de nuestras aplicaciones. Los MRO crean un modelo del programa orientado a objetos con un alto nivel de abstracción. En otras palabras, crea un nivel de lógica sin los detalles subyacentes del código. El mapeo describe la relación entre un objeto y los datos sin saber cómo están estructurados los datos [3,4]. Luego, el modelo se puede usar para conectar la aplicación con el código SQL necesario para administrar las actividades de datos.

Este tipo de código de con su alto nivel de abstracción no tiene que volver a escribirse, lo que le ahorra al desarrollador una enorme cantidad de tiempo.

En la figura 2 se pueden observar 3 actores principales, el lenguaje (Java) que implementa el modelo por medio de clases encapsuladas, una herramienta de mapeo (Hibernate) que facilita el mapeo de datos y un esquema de base de datos para almacenar los registros.

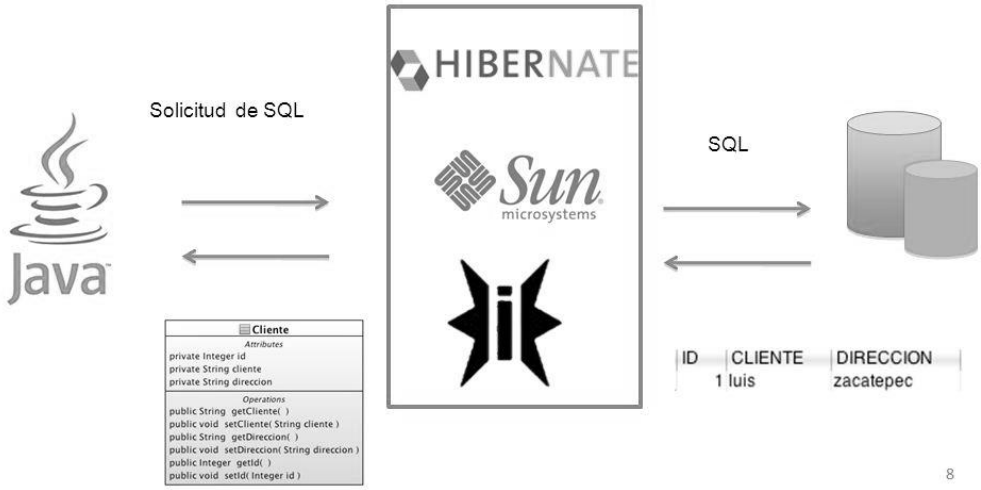

Figura 2. Diagrama representativo de los actores del Mapeo Relacional de Objetos.

Los MRO emplean dos estrategias diferentes:

1. patrón de registro activo
2. patrón de mapeador de datos.

Patrón de registro activo. Esta estrategia asigna datos dentro de la estructura de los objetos en el código. Maneja datos usando clases y estructuras dentro de tu código de programación. Este método tiene problemas ya que la estructura de la base de datos está estrechamente conectada con el código, lo que dificulta eliminar la base de datos y migrarla a una aplicación diferente.

Patrones de mapeador de datos. El patrón del mapeador de datos intenta desacoplar la lógica comercial en los objetos de la base de datos. Esta separación puede facilitar el cambio de bases de datos y utilizar la misma lógica de programación.

3. ANÁLISIS EL USO DE MRO's

Los desarrolladores pueden usar código SQL sin procesar para escribir una interfaz directa entre la aplicación y la base de datos. La mayoría de las bases de datos relacionales admiten SQL para crear aplicaciones e interfaces de datos. Es estable y, dado que SQL se ha utilizado desde la década de 1970, está bien documentado y respaldado. Los programadores mantienen mucho control sobre su interfaz de datos con SQL. Requiere mucho trabajo, pero es más flexible y detallado que una abstracción MRO.

Las herramientas de MRO son populares entre los desarrolladores de Programación Orientada a Objetos porque minimizan la cantidad de conocimientos de SQL necesarios para conectar una base de datos a una aplicación. Los MRO también generan automáticamente el código SQL, lo que le permite concentrarse en generar la lógica comercial. Hay cuatro beneficios significativos de usar mapeadores relacionales con objetos para administrar la interfaz entre aplicaciones y bases de datos.

1. **Productividad.** Escribir código de acceso a datos requiere mucho tiempo y no agrega mucho valor a la funcionalidad de la aplicación. Es esencialmente la plomería del código. El uso de una herramienta como un MRO que genera automáticamente el código de acceso a los datos, ahorra un tiempo de desarrollo tremendo que no agrega valor a la aplicación. En algunos casos, el MRO puede escribir el 100 por ciento del código de acceso a datos para la aplicación. El MRO también puede ayudar~~lo~~ a realizar un seguimiento de los cambios en la base de datos, lo que facilita la depuración y el cambio de la aplicación en el futuro.

2. **Diseño de aplicaciones.** Un MRO bien escrito implementará patrones de diseño para obligarlo a utilizar las mejores prácticas para el diseño de aplicaciones. Si usa un MRO para administrar la interfaz de datos, no necesita crear el esquema de base de datos perfecto de antemano. Podrá cambiar la interfaz existente fácilmente. Separar la tabla de la base de datos del código de programación también le permite cambiar los datos para diferentes aplicaciones.

3. **Reutilización de código.** Una forma de reutilizar los datos es crear una biblioteca de clases para generar una biblioteca de vínculos dinámicos (DLL) separada. Puede crear una nueva aplicación sin necesidad de duplicar el código de acceso a datos.

4. **Pruebas reducidas.** Dado que el código generado por el MRO está bien probado, no es necesario que dedique tanto tiempo a probar el código de acceso a datos. En su lugar, puede concentrarse en probar la lógica y el código de negocios.

Los MRO son una excelente herramienta para muchas aplicaciones, pero algunos desarrolladores encontraron varios inconvenientes al usar MRO para aplicaciones de acceso a datos. Los problemas parecen estar relacionados con la complejidad de la aplicación. Con aplicaciones simples, tener un alto nivel de abstracción ayuda al proceso de desarrollo. Pero cuando las aplicaciones son complejas, la abstracción cubre muchos detalles necesarios para abordar problemas relacionados con los datos [5].

a) **Actuación.** Una queja común entre los desarrolladores de programación orientada a objetos es el código extra generado por el MRO. El código agregado ralentiza el rendimiento de la aplicación y dificulta su mantenimiento. Un MRO bien diseñado debería poder crear código de alta calidad sin afectar la velocidad de la aplicación.

b) **Necesidad de saber SQL.** Las abstracciones de alto nivel no siempre generan el mejor código SQL y los desarrolladores no pueden confiar en el MRO el 100 por ciento del tiempo. Aún necesita conocer SQL y la sintaxis generada por el MRO.

c) **Mapeo deficiente.** Los MRO a veces pueden crear un mapeo incorrecto entre las tablas de datos y los objetos. Estos problemas pueden causar problemas de aplicación y ser difíciles de reconocer. Los MRO

también fomentan el mapeo uno a uno, aunque es raro que las aplicaciones comerciales tengan muchas relaciones uno a uno.

d) **Efectos en el esquema y las migraciones.** Una capa MRO mal escrita a menudo dificulta la mejora de los esquemas de datos. A menudo puede limitar sus opciones y, según el MRO, sus opciones pueden ser limitadas. Si su MRO no admite migraciones, escribir las migraciones en su POO es más complicado que escribir el código para las migraciones en SQL. Un MRO bien escrito incluirá herramientas para facilitar futuras migraciones de bases de datos.

4. DISCUSIÓN Y CONCLUSIONES

Escribir código SQL para adjuntar una base de datos relacional a una aplicación orientada a objetos puede ser una actividad que requiere mucho tiempo y genera poco valor para la aplicación comercial. Los desarrolladores pueden escribir código SQL sin procesar o usar generadores de consultas SQL para mejorar el proceso, pero ambos métodos aún requieren un conocimiento profundo de la base de datos y la capacidad de codificar en SQL. Los MRO mejoran la productividad mediante la creación de modelos de datos altamente abstractos y la generación automática de código SQL. Estas herramientas también facilitan la separación de la base de datos de la lógica de programación, lo que brinda a los desarrolladores más flexibilidad. Las quejas comunes incluyen rendimiento reducido, codificación adicional y mapeo deficiente según la calidad de MRO. Los desarrolladores deben hacer su tarea antes de comprometerse a usar un MRO para acelerar el proceso de desarrollo de programación orientada a objetos. Hay varias herramientas comerciales y de código abierto disponibles. Dado que estas herramientas han estado disponibles durante muchos años, están bien documentadas y respaldadas por extensas comunidades de desarrollo. Sin embargo, las implementaciones de estas herramientas son propias del tamaño del proyecto, tiempo de entrega, capacidades del equipo de desarrollo y presupuesto; todas estos son factores que influirán en el proceso de selección del lenguaje, librería, framework y gestor de base de datos.

REFERENCIAS

[1] P. J. Deitel, H. M. Deitel, and A. Vidal Romero Elizondo, *Java: como programar*. Pearson educacion, 2008.
[2] G. S. Linoff, *Data Analysis Using SQL and Excel*, 1st Edition. Indianapolis: Wiley Publising Inc., 2008.
[3] Shabda Raaj and Yash Rastogi (Author), "Django ORM Cookbook Documentation Release 2.0 Agiliq," 2019.
[4] C. Bauer, G. King, G. Gregory, and L. Demichiel, *Java persistence with hibernate*, 2nd Edition. 2016. [Online]. Available: www.allitebooks.com
[5] Mario Hoyos, "What is an ORM and Why You Should Use it," *https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a*, Dec. 24, 2018.

Correo de autor: ***miguels.gomez.diaz@gmail.com***