Mapeamento Objeto-Relacional – Considerações sobre a Ferramenta Ponte entre Programação Orientada a Objetos e Base de Dados Relacional

Maria do Carmo Nicoletti
PMCC-UNIFACCAMP
Campo Limpo Paulista, SP, Brasil
carmo@cc.faccamp.br

Denilson Silva Marçal
PMCC-UNIFACCAMP
Campo Limpo Paulista, SP, Brasil
denilson.marcal.professor@gmail.com

RESUMO

Este artigo tem por foco a apresentação e discussão de uma ferramenta de software conhecida como Mapeamento Objeto-Relacional (ORM) abordada por meio de quatro de seus aspectos relevantes: (1) apresentação e caracterização do que é considerado um Mapeamento Objeto-Relacional em contextos de sistemas computacionais que empregam Programação Orientada a Objetos (OOP) e Base de Dados Relacional (RDB); (2) comentar brevemente sobre a origem e as justificativas para a criação e uso de ORMs; (3) discutir os ambientes computacionais em que ORMs são necessários e, eventualmente, são empregados e (4) considerar as principais vantagens e desvantagens no uso de ORMs em ambientes computacionais, com relação ao desempenho computacional de inúmeros sistemas computacionais ativos.

ABSTRACT

This article focuses on the presentation and discussion of a software tool known as Object-Relational Mapping (ORM) approached through four of its relevant aspects: (1) presentation and characterization of what is considered an Object-Relational Mapping (ORM) in a context of computing systems that employ object-oriented programming (OOP) and relational databases (RDB); (2) briefly comment on the origin and justifications for the creation and use of ORMs; (3) discuss computational environments in which ORMs are necessary and, eventually, are employed and (4) consider the main advantages and disadvantages when using ORMs in computational environments, in relation to the computational performance of numerous active computational systems.

1. Contextualização e Considerações Iniciais

Em um contexto computacional as chamadas Bases de Dados (DBs) podem ser abstraídas como estruturas computacionais lógicas que organizam e armazenam dados com vários objetivos e, particularmente, para torná-los passíveis de serem gerenciados, recuperados e continuamente usados e atualizados. Em ambientes computacionais que fazem uso de DBs, essas bases usualmente são gerenciadas por um programa que é referenciado como Sistema de Gerenciamento de Base de Dados (SGBD). É esperado que SGBDs forneçam interfaces lógicas para usuários e para programas aplicativos acessarem e administrarem parte dos dados. BDs podem ser agrupadas em dois grandes grupos:

(1) as identificadas como Base de Dados Relacional (RDB), em que dados são organizados e modelados para serem percebidos por usuários como um conjunto de tabelas, em que cada tabela é referenciada como relação. O modelo relacional foi proposto em 1970 por E. F. Codd [Codd 1970] e novamente abordado e discutido em [Codd 1985a] [Codd 1985b]. Dentre as bases de dados relacionais mais populares estão SQL Server (Microsoft) [RDB_1], Oracle [RDB_2] e MySql [RDB_3].

(2) as identificadas como Base de Dados Não Relacional (NoSQL), que surgiram por volta do ano 2000 e que viabilizam a possibilidade de dados relacionados estarem organizados em apenas uma estrutura de dados o que, de certa forma, contribui para a redução de custos de armazenamento. De acordo com comentários e explicações de desenvolvedores que utilizam

NoSQL [NoSQL 2024], tais DBs têm capacidade de responder a situações não planejadas promovendo com isso, o desenvolvimento de aplicativos com maior agilidade e rapidez. Como informado em [NoSQL5 2024] os cinco NoSQLs mais populares são: MongoDB [$NoSQL_1$], AmazonDynamoDB [$NoSQL_2$], DataStax [$NoSQL_3$], Couchbase [$NoSQL_4$] e ElasticSearch [$NoSQL_5$].

O avanço na adoção de programação orientada a objetos (OOP) em ambientes computacionais que disponibilizam RDBs e, também, as dificuldades técnicas que surgiram quando da articulação entre sistemas codificados usando OOP e RDBs, impulsionaram o desenvolvimento de softwares caracterizados como Mapeamento Objeto-Relacional (ORM), considerados intermediadores e facilitadores da comunicação entre sistemas desenvolvidos usando OOP e RDBs.

2. Recuperação Parcial da História de ORMs

A estrutura ORM teve origem no contexto de duas tendências significativas na área de desenvolvimento de software: programação orientada a objetos e base de dados relacional. Como informado em [Rickerby 2015], à medida que o custo do hardware comum começou a declinar a partir de meados da década de 1980, a indústria começou a abandonar os computadores conhecidos como *mainframes* e passou a adotar a implantação em massa de aplicativos GUI (*Graphical User Interface*), termo utilizado para descrever a interface gráfica de um software ou aplicativo, que permite que usuários possam interagir com um sistema computacional via computador, *tablet*, celular ou qualquer outro dispositivo eletrônico baseado em um modelo cliente-servidor.

À época, entretanto, o mundo empresarial relacionado a negócios ainda dependia de *mainframes* e das bases de dados relacional que surgiram nas décadas de 1970 e 1980, capazes de armazenar enormes volumes de informações persistentes, e que já tinham alcançado certa estabilidade bem, como atingido um alto grau de confiabilidade por parte da comunidade de usuários.

Como Rickerby comenta em [Rickerby 2015], uma das principais responsabilidades desta classe emergente de "aplicações empresariais" era conectar-se a estas bases de dados, liberando o potencial do software para automatizar fluxos tediosos de trabalho, bem como permitir e capacitar que utilizadores não técnicos pudessem gerenciar e manipular dados. Neste contexto de transição a programação orientada a objetos acabou se estabelecendo amplamente tanto na indústria quanto em pesquisas acadêmicas.

À medida que os ideais altamente divulgados da programação orientada a objetos foram entrando em conflito com a bem estabelecida e comprovada tecnologia de base de dados relacional um problema, caracterizado como "*incompatibilidade de impedância objeto-relacional*", que antes inexistia, emergiu e se tornou cada vez mais presente. O conceito de impedância, importado da área de engenharia elétrica, foi usado para caracterizar problemas de incompatibilidades que surgiam quando códigos orientados a objetos trocavam dados com base de dados relacional. Problemas relacionados a essas incompatibilidades foram abordados com a introdução de uma camada de mapeamento entre programação orientada a objetos e bases de dados relacional. Teixeira, em [Teixeira 2017], informa que a opção mais usada para a redução do atrito entre código orientado a objetos e base de dados relacional seria por meio do uso de *frameworks*. Alguns desses *frameworks* são abordados na Seção 4.

3. Contextualização e Considerações Iniciais sobre ORMs

No cenário atual de desenvolvimento de software, o uso de sistemas de gerenciamento de bas de dados relacional para persistir objetos de um modelo de domínio orientado a objetos é uma abordagem comum que é adotada por um vasto número de organizações.

Como pode ser encontrado em inúmeras referências com foco em desenvolvimento de software, o termo ORM (*Object-Relational Mapping*) [Ambler 1997] pode ser caracterizado

como uma estratégia técnica, muitas vezes identificada como uma ferramenta computacional, que tem como objetivo primeiro o de adequar o modelo de Desenvolvimento de Software Orientado a Objetos (*i.e.*, uso de linguagens de programação baseada em objetos (OOP) no desenvolvimento de sistemas computacionais) ao modelo de Base de Dados Relacional (RDB). Em inúmeras publicações técnicas/acadêmicas a necessidade de uma 'ponte' entre OOP e RDB é enfatizada e várias dessas publicações insistem que a necessidade de tal 'ponte' é consequência do processo de desenvolvimento de software orientado a objetos ser uma proposta bem mais recente, quando comparada com a época (a partir dos anos 70) em que RDBs surgiram, se estabeleceram com grande sucesso e continuam em uso até os dias atuais. Nas discussões e comentários encontrados em [Lorenz *et al.*, 2016] ORM é abordado e definido como um mecanismo desenvolvido com o objetivo de preencher a lacuna semântica entre linguagens de programação orientada a objetos (OOP) e SGBDs.

No contexto de muitos dos ambientes organizacionais de desenvolvimento de software atuais, ORMs estão sendo frequentemente desenvolvidas e disponibilizadas, com o objetivo de facilitar a acomodação de programação baseada em objetos a ambientes computacionais que fazem uso de RDBs. Como argumentado e justificado em [Barnes 2007] ORM é uma tecnologia que busca automatizar a conexão entre o ambiente orientado a objetos e o ambiente relacional, minimizando a duplicação de dados, o custo de manutenção e a suscetibilidade a erros que o estabelecimento da conexão pode introduzir. Presentemente, um grande número e uma variedade de ferramentas ORM estão disponibilizados no mercado; muitas delas, entretanto, estão ainda em fase de desenvolvimento. É fato, entretanto, que várias ferramentas ORM disponibilizadas não promovem uma persistência de dados totalmente transparente e, talvez devido a isso, não tem sido muito populares na indústria de software.

Devido aos conceitos fundamentalmente diferentes empregados por ambos, a saber, álgebra relacional (empregada em RDBs) e o modelo orientado a objetos incorporado às linguagens de programação, um conjunto de desafios afloram quando do mapeamento entre objetos e tabelas (relações em RDBs). Apesar das diferenças entre as estruturas existentes serem muitas, todas elas, de certa forma, traduzem a mesma ideia, que é a de fornecer um mapeamento entre a camada de objetos (classe object) e relações (tabelas da RDB), que são dois conceitos que diferem substancialmente entre si. Objetos são temporários enquanto dados no modelo relacional são persistentes. Objetos não obedecem a um esquema restrito, enquanto a linguagem de consulta estruturada (SQL) de RDBs segue um esquema restrito. Em [Sivakumar *et al.* 2021] os autores comentam que uma estrutura de mapeamento objeto-relacional atravessa a ponte da diferença entre objeto e relação, fornecendo uma interface para a linguagem de programação. Essa interface pode ser usada para permitir que a estrutura de mapeamento objeto-relacional gere consultas SQL e, portanto, a aplicação *per se* não precisa conter código SQL e tampouco ter recursos SQL.

Como uma estrutura ORM automaticamente gera consultas à RDB, o desenvolvedor acaba exercendo pouco controle sobre a qualidade e eficiência de tais consultas. Os autores em [Sivakumar *et al.* 2021] lembram que se o sistema de software sendo desenvolvido é volumoso, pode acontecer do controle do desenvolvedor sobre as consultas geradas ser menor ainda. Na discussão sobre ferramentas computacionais que automaticamente produzem implementações, apresentada em [Bagheri *et al.* 2017], os autores comentam que tais ferramentas frequentemente usam estratégias de *ponto único* (ver [Kaye 2000]). Na discussão que os autores apresentam, dentre as várias ferramentas listadas, estão incluídas as ferramentas ORM que são disponibilizadas em muitos ambientes de programação, relembrando que ferramentas caracterizadas como objeto-relacional mapeiam modelos de dados orientados a objetos para esquemas relacionais e geram código para gerenciar dados do aplicativo de software.

Ferramentas ORM usualmente contemplam apenas uma única estratégia de mapeamento e não colaboram para que desenvolvedores entendam as soluções disponíveis. É importante comentar que, na prática, a análise sistemática do espaço abrangido pelo mapeamento é muitas vezes impraticável, devido ao número de possibilidades. O processo de implementação de uma estrutura ORM está atrelado a um conjunto de problemas/tarefas a serem resolvidos/executadas,

tais como efetividade, consistência de dados e facilidade de uso da estrutura por desenvolvedores. Obviamente a programação baseada em objetos depende da linguagem computacional sendo utilizada.

É fato que levantamentos bibliográficos no universo de literatura técnica/acadêmica com foco no uso de ORMs têm se tornado cada vez mais comum na indústria de software; tais levantamentos evidenciam que o número de adoções de ORMs está aumentando. A razão mais simples para esse aumento se deve ao fato que o emprego de ORMs simplifica a interação com RDBs, permitindo também que desenvolvedores de software trabalhem com linguagens de programação orientadas a objetos ao invés de tabelas e consultas SQL. No entanto, ORMs considerados tradicionais muitas vezes enfrentam problemas relacionados a desempenho e consumo excessivo de memória, particularmente em cenários que empregam grandes volumes de dados [Bagheri *et al.* 2017].

4. Situações Enfrentadas em Ambientes Computacionais com ORM

Presentemente, um dos problemas mais comuns enfrentados por desenvolvedores de software está relacionado à busca da otimização de desempenho do produto desenvolvido, em ambientes computacionis que envolvem o uso da estrutura ORM. Embora ORMs forneçam abstrações convenientes para interações com RDBs, muitas vezes consultas mal elaboradas, ineficientes, bem como aquelas envolvendo um número excessivo de objetos podem impactar o desempenho. Não pode ser esquecido que o ORM pode criar *queries* SQL razoavelmente complexas que podem não ser conveniente para a RDB disponível.

Apesar de ser um conhecimento trivial e básico, desenvolvedores podem ser descuidados ao projetar e otimizar consultas e, muitas vezes, se esquecem de considerar os mecanismos de cache e de ajuste das configurações do ORM, com o objetivo de garantir acesso eficiente aos dados e minimizar a sobrecarga. Uma sugestão a desenvolvedores que pode ser encontrada em várias publicações envolvendo ORM (*e.g.* [Procaccianti *et al.* 2016]) é a de que busquem um equilíbrio entre a conveniência dos recursos disponibilizados por ORM e os aspectos relacionados a desempenho, uma vez que requerem planejamento e otimizaçao ao longo de todo o processo de desenvolvimento.

A manutenção da integridade e da consistência dos dados em um ambiente computacional com diferentes DBs pode, eventualmente, se tornar uma tarefa difícil para desenvolvedores em um ambiente com ORM, uma vez que detalhes e restrições nas RDBs, tais como chaves estrangeiras, podem não ser levadas em consideração, o que pode provocar anomalias nos dados, usualmente erros ou conflitos, se o ORM não usar as mesmas regras usadas nos RDBs. Para contornar esses problemas, desenvolvedores que trabalham sob o ORM precisam recorrer ao uso de transações, bloqueios e mecanismos de controle de simultaneidade, bem como sincronizar regularmente ORM e a RDB.

O aprendizado e o domínio da conceituação e dos processos envolvidos na ferramenta ORM são de fundamental importância para que desenvolvedores não familiarizados com a ferramenta adquiram conhecimento especializado, por meio da fixação dos conceitos relevantes relacionados a OOP e ORM, com o objetivo de promover bom desempenho e agilidade de desenvolvedores inexperientes.

Apesar de quase sempre ser relegada a segundo plano na maioria de ambientes computacionais, a disponibilização de documentação atualizada sobre os ambientes computacionais em geral e, particularmente, aqueles envolvendo ORM é um importante aspecto a ser cuidado. Um aspecto que deve ser sempre considerado por desenvolvedores é o de garantir que a ferramenta ORM seja compatível e portátil entre diferentes RDBs e plataformas. Eventualmente ORMs podem ter diferenças com relação aos vários recursos associados a DBs, tais como tipo de dados, funções disponibilizadas, operadores, bem como conflitos com outras bibliotecas. Uma boa prática que desenvolvedores devem investir para evitar problemas é a de investir em verificações e testes relacionados à compatibilidade e portabilidade da ferramenta com DBs.

Uma preocupação constante de desenvolvedores de ORM é a do projeto e arquitetura da camada ORM, bem como a de estabelecer a lógica de acesso a dados que contempla sustentabilidade, escalabilidade e reutilização. O uso de ORM pode afetar ambos, o projeto e a arquitetura de aplicações como, por exemplo, a escolha do modelo de dados, o projeto direcionado a domínio, o padrão do repositório, a camada de serviço ou o padrão de unidade de trabalho. ORM também pode introduzir alguns conflitos entre projeto e arquitetura, como incompatibilidade de impedância, carregamento lento, mapa de identidade ou padrão de registro ativo.

Quando do desenvolvimento de ORMs, o processo de teste envolve a validação de consultas, operações de dados e gerenciamento de transações, por meio de testes unitários e de integração. A depuração envolve rastrear fluxos de execução, inspecionar consultas SQL e monitorar interações de base de dados para identificar problemas tais como mapeamentos incorretos ou consultas ineficientes. Ferramentas de depuração e registro específicos de ORM ajudam a diagnosticar e resolver problemas de maneira eficaz durante o desenvolvimento e a manutenção. Testar o código ORM pode ser complicado devido à complexidade das interações entre o código e a base de dados. Desenvolver testes de unidade que simulam interações de base de dados e depurar problemas quando um ORM abstrai a SQL pode ser demorado e exigir um conhecimento profundo do ORM e da base de dados em questão.

Para lidar com muitas situações indesejáveis, desenvolvedores de ORM deveriam seguir os princípios agrupados no SOLID [Martin 2024], que é um acronico composto pelas iniciais de cinco princípios, cada um deles associado a uma prática para desenvolvimento de software. Como informa o seu criador, a adoção dessas práticas pode evitar problemas. Os princípios estabelecem práticas para o desenvolvimento de software com projeções para mantê-lo, bem como para ampliá-lo, à medida que o projeto se desenvolve. Os princípios são: **S**: *Single-responsibility Principle*, **O**: *Open-closed Principle*, **L**: *Liskov Substitution Principle*, **I**: *Interface Segregation Principle* e **D**: *Dependency Inversion Principle*. A descrição de cada um dos princípios pode ser encontrada no *link* mostrado na referência [Martin 2024].

5. Considerações Finais

É importante notar que ao longo do levantamento bibliográfico que tem sido conduzido desde o início de um projeto de pesquisa, do qual esse artigo é *spin-off*, ficou óbvia a inexistência de um consenso com relação à terminologia adotada em muitos dos trabalhos de pesquisa investigados relacionados à ORM. Em muitos deles uma variedade de palavras sinônimas são usadas para nomear referências a um mesmo conceito, fato que contribui para promover ambiguidade e compreensão equivocada entre conceitos e procedimentos utilizados.

Também, na literatura é comum encontrar definições de conceitos que são feitas via apresentação de exemplos apenas, quando deveriam ter sido formalmente apresentadas, empregando vocabulário padronizado dos conceitos apresentados. Esse problema foi também detectado e comentado pelos autores do artigo [Lorenz *et al.* 2016], que enfatizam que a falta de padronização e de definições formais podem provocar o surgimento de inconsistências.

Referências

[Ambler 1997] Ambler S. W. (1997) Mapping objects to relational databases, An AmbySoft Inc. White Paper, http://www.AmbySoft.com/mappingObjects.pdf

[Bagheri *et al.* 2017] Bagheri, H.; Tang, C.; Sullivan, K. (2017) Automated synthesis and dynamic analysis of tradeoff spaces for object-relational mapping, *IEEE Transactions on Software Engineering*, v. 43, no. 2, pp. 1-20, Digital Object Identifier no. 10.1109/TSE.2016.258764.

[BDR_1 2024] https://en.wikipedia.org/wiki/Microsoft_SQL_Server

[BDR_2 2024] https://en.wikipedia.org/wiki/Oracle_Database

[BDR_3 2024] https://en.wikipedia.org/wiki/MySQL

[Codd 1970] Codd, EF (1970) A relational model of data for large shared data banks, *Communications of the ACM* , v.**13**, no. 6, pp. 377–387, doi:10.1145/362384.362685.S2CID207549016

[Codd 1985a] Codd, E. F. (1985) Is your DBMS really relational?, *Computerworld*, October 14th, v. 19, pID1, consultado em 7-04-2024.

[Codd 1985b] Codd, E. F. (1985) Does your DBMS run by the rules? (1985) *Computerworld*, October 21st, v. 19, p. 49, consultado em 20-04-2024.

[Kaye 2000] Kaye, R. (2000). Minesweeper is NP−Complete, *Mathematical Intelligencer*, v. 22, no. 2, pp. 9−15.

[Lorenz *et al* 2017] Lorenz, M; Rudolph, JP; Hesse, G; Uflacker, M; Plattner, H (2017) Object-relational mapping revisited – a quantitative study on the impact of database technology on O/R mapping strategies, In: Proc. of Hawaii International Conference on System Sciences (HICSS), pp. 4877-4886 (DOI:10.24251/hicss.2017.592)

[Martin 2024] Martin, RC (2024) First five object-oriented design principles, https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design, consultado em 30-05-2024)

[NoSQL 2024] https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-nosql-database/

[NoSQL5 2024] https://www.kron.digital/5-bancos-de-dados-nosql-populares-que-todo-profissional-de-big-data-deve-conhecer/

[$NoSQL_1$2024] https://en.wikipedia.org/wiki/MongoDB

[$NoSQL_2$ 2024] https://en.wikipedia.org/wiki/Amazon_DynamoDB

[$NoSQL_3$ 2024] https://en.wikipedia.org/wiki/DataStax

[$NoSQL_4$ 2024] https://en.wikipedia.org/wiki/Couchbase_Server

[$NoSQL_5$ 2024] https://en.wikipedia.org/wiki/Elasticsearc

[Procaccianti *et al.* 2016] Procaccianti G, Lago P, Diesveld W (2016) Energy Efficiency of ORM Approaches: an Empirical Evaluation, In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2016, pp. 1-10.

[Rickerby 2015] Rickerby, M. (2015) The rise and fall of object relational mapping, https://maetl.net/talks/rise-and-fall-of-orm consultado 13-05-2024

[Sivakumar *et al.* 2021] Sivakumar, V.; Balachander, T.; Logu, M.; Jannali, R. (2021) Object relational mapping framework performance impact, *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, v. 12, no. 7, pp. 2516–2519. Retrieved from https://turcomat.org/index.php/turkbilmat/article/view/3580

[Teixeira 2017] Teixeira, M. H. (2017) Impedância objeto relacional - O atrito natural entre os dois mundos, *Tecnologia em Projeção*, vol. 8, no. 1, pp. 11-20.