Unisanta Science and Technology, **2023, November**
Published Online 2023 Vol. 12 nº2 http://periodicos.unisanta.br/index.php/sat

UNISANTA
Science and Technology
ISSN 2317-1316

ESTUDO DA APLICABILIDADE DE SERIALIZAÇÃO DE OBJETOS NA PERSISTÊNCIA DE DADOS EM SISTEMAS DE INFORMAÇÃO

Helio Augusto de Lima Rangel, Nina Maria Bueno, Ana Carolina Caetano Senger, Antonio Carlos Marques do Amaral Guerra, Claudio Ferreira de Carvalho, Claudio Souza Nunes, Maurício Neves Asenjo

UNISANTA– Universidade Santa Cecília – Faculdade de Ciências Exatas, Engenharia e Arquitetura – Bacharelado em Sistemas de Informação e Tecnólogo em Análise e Desenvolvimento de Sistemas
Rua Oswaldo Cruz, 277 - Santos-SP, Brasil - CEP: 11045-907

E-mail*:* hlrangel@unisanta.br
Received September, 2023

Resumo: Este artigo procura avaliar as situações em que o uso de serialização de objetos pode ser justificado no lugar do uso de um banco de dados relacional. Para isso, vamos comparar os recursos típicos de um banco de dados, principalmente a normalização, com a integridade dos objetos em memória e analisar as soluções de sistemas híbridos, compartilhando as duas tecnologias de persistência no que cada uma tem a oferecer.

Palavras-chave: Bancos de Dados Relacionais, Persistência, Serialização

STUDY OF THE APPLICABILITY OF OBJECT SERIALIZATION IN DATA PERSISTENCE IN OO INFORMATION SYSTEMS

Abstract: This article seeks to evaluate situations where the use of object serialization can be justified in place of the use of a relational database. To do this, we will compare the typical features of a database, normalization, with the integrity of objects in memory and analyze hybrid systems solutions, sharing the two persistence technologies in terms of what each one has to offer.

Keywords: Persistence, Relational Databases, Serialization

1. Introdução

Quando pensamos em persistência de dados, ou seja, salvar e organizar os dados em memória, a primeira ideia que nos ocorre é o uso de um banco de dados relacional, o que é plenamente justificado, uma vez que eles evoluíram para atender demandas relativas aos dados como: segurança; facilidade de acesso, atualização, inserção e deleção; criação e alteração de tabelas; criação e atualização de índices, chaves; criação e uso de *stored procedures;* criação e uso de *triggers;* criação e uso de *views;* transações etc.

O uso de persistência, em geral, se justifica principalmente pelo fato de que em sistemas executados em computadores e servidores tradicionais, os dados são trabalhados em memória RAM que, por se perderem ao serem desligados, precisam se manter seguros e atualizados em alguma memória não volátil.

Em sistemas desenvolvidos com paradigma de orientação a objetos, a maneira como em um banco relacional distribuímos os dados em tabelas diferentes, nos força a "explodir" os atributos dos objetos em diversas tabelas para que atendam os princípios da normalização. Isso não é por si só um impedimento para utilização de um banco relacional, mas reconhecemos que alguns benefícios do Paradigma de Orientação a Objetos serão perdidos. Existem disponíveis no mercado muitos banco de dados Orientado a Objetos, mas, por algum motivo, não são muito populares. Outra razão para optar por bancos relacionais é a frequente necessidade de

Copyright © 2024 ISESC

compartilhamento de dados em tempo real com outros sistemas legados.

Neste artigo, discutimos os casos em que o uso de serialização de objetos na persistência, parcial ou total pode se justificar, trazendo benefício de agilidade no desenvolvimento, sem perda de segurança nem integridade dos dados.

2. Desenvolvimento

2.1. A Orientação a Objetos (OO)

> Sob o ponto de vista da linha do tempo, não podemos dizer que a orientação a objetos é uma abordagem nova ou recente. Os primeiros registros que se tem notícia daquilo que veio a se tornar a orientação a objetos, datam do final da década de 60, com o surgimento de uma linguagem chamada Simula-68. Como podemos imaginar, o nascimento ainda incipiente foi aprimorado durante a década seguinte, principalmente em conjunto com outra linguagem, o Smalltalk. O amadurecimento e popularização, de fato, deu-se ao longo das décadas de 80 e 90, novamente em conjunto com outras linguagens de programação, dessa vez, o Java e o C++. (VERSOLATTO, 2023)

Orientação a Objetos é um paradigma de desenvolvimento de sistemas, baseado em classes e objetos que podem definir dados conhecidos como atributos e código que são os procedimentos, conhecidos como métodos. Um dos principais objetivos da Orientação a Objetos é permitir a reutilização de código, tornando os sistemas de *software* mais modulares e manuteníveis. (HENRIQUE, 2023)

Pensar orientado a objetos, para a maioria das pessoas, é um desafio. Nosso cérebro foi moldado pela evolução para pensar de forma procedural e, por outro lado, o pensamento OO exige um alto grau de abstração. Devagar, o pensamento OO foi tomando conta do mundo da programação e linguagens como C++, Java e C# se tornaram cada vez mais populares. Ainda hoje se vê alguma resistência ao paradigma da Orientação a Objetos, mesmo assim praticamente todas as linguagens modernas suportam OO, mesmo que, em alguns casos, falte algum dos conceitos básicos que são: Classes; Objetos; Encapsulamento; Herança; Polimorfismo. Mesmo com todo o sucesso e popularidade da OO, ela não é unanimidade no mundo da programação. Muitos, por um motivo ou outro, não se sentem à vontade com ela. (FÉLIX, 2024)

2.2. A Serialização de Objetos

A serialização acontece convertendo um objeto em memória, em uma sequência de bytes de tal forma que possa ser manipulado. Após a transmissão ou o armazenamento esta cadeia de bytes pode ser transformada novamente no objeto que o originou. A ideia por trás da serialização é a de "congelar" o objeto, guardando-o por um tempo indeterminado, movê-lo e depois, quando for o momento adequado, "descongelar" esse objeto tornando novamente utilizável.

Apenas atributos de instância de uma classe são serializados, não incluindo os atributos estáticos. Se o objeto a ser serializado for proveniente de uma subclasse, todos os atributos de instância, mesmos aqueles originários de superclasses, serão serializados também. Se um objeto fizer referência a outros objetos, todos seus atributos serão também serializados garantindo assim a sua integridade. (BERTOL, 2012)

Um problema da serialização é que se, no intervalo de uma serialização de um objeto e a sua desserialização a classe a que o objeto pertence pode sofrer alteração nos seus atributos, fazendo com que a recuperação do objeto se torne inviável. As linguagens OO oferecem alternativas. No caso do Java, por exemplo, temos o *SerialVersionUID* que é um número que identifica a versão da classe que foi usada durante o processo de serialização. Esse número é utilizado para rastrear a compatibilidade de versões serializadas das classes e saber se o objeto que se está recuperando é de uma versão “compatível”. (SILVA, 2017)

2.3. Como a persistência via serialização funciona

Basicamente estamos preocupados com a recuperação dos dados armazenados em memória RAM no caso de perda de alimentação elétrica. Enquanto os objetos estiverem em memória, os arquivos contendo os objetos serializados não terão utilidade, são basicamente um *back-up*. Os arquivos de objetos serializados serão finalmente úteis no momento que o sistema “sobe” para a memória, ou seja, ele é executado, reestabelecendo exatamente o estado dos objetos do momento em que o sistema caiu ou foi desligado. (PREISSLER, 2012)

2.4. Um exemplo de persistência via serialização

Podemos imaginar um sistema de faturamento com lista de clientes, lista de produtos e lista de fornecedores e, a partir destas três listas, gerar uma quarta de faturas. Cada novo objeto desta lista precisa referenciar clientes, produtos e fornecedores. Estas referências, enquanto em memória, serão sempre corretas, mesmo que se faça alterações nos objetos referenciados. Pensando em manter um

Copyright © 2023 ISESC

espelho atualizado do estado dos objetos em memória, sempre que um objeto sofrer alguma alteração, todos devem ser serializadas ao "mesmo tempo", sem que nenhuma alteração seja feita em qualquer um deles durante a serialização. O motivo de não serializar apenas o objeto que está sofrendo alteração é que trabalhamos com os objetos estruturados e organizados em listas e vale a pena tratar a lista ao invés de cada objeto individualmente; (PREISSLER, 2012)

No momento da carga do sistema, todas as listas: clientes; produtos; fornecedores e faturas devem ser carregados para memória (desserializadas) e, durante o funcionamento do sistema, a cada nova fatura ou atualização, a lista de faturas deve ser serializada e salva imediatamente, mantendo o arquivo serializado sempre sincronizado com os objetos em memória. A necessidade de manter todos os objetos em memória, no caso de sistemas com muitos registros, pode comprometer a capacidade de memória utilizada pela aplicação.

2.5. Traçando um paralelo entre as técnicas de persistência

No caso de persistência utilizando banco de dados relacional, a busca e carga do objeto Fatura, por exemplo, precisa acessar todas as tabelas referenciadas via chave estrangeira da fatura. No caso da persistência via serialização, o objeto Fatura está disponível em memória. No caso de persistência via banco de dados, quando realizamos atualizações, o banco de dados terá apenas que processar os registros que foram efetivamente modificados, representando uma carga bem menor de processamento.

2.5.1. Projetando um sistema OO para persistência via Banco Relacional

O trabalho de modelagem do banco de dados é crucial para o sucesso do projeto. O maior ou menor cuidado nesta fase pode ser responsável pelo sucesso ou fracasso e, por este motivo, procuramos fazer uma análise completa de todas as classes envolvidas e, a partir delas, modelamos o banco projetando tabelas capazes de representar da melhor maneira possível as classes envolvidas. No projeto do banco, talvez a parte mais crítica seja a fase de normalização. A normalização estabelece a melhor maneira de se distribuir os atributos das classes nas diversas tabelas envolvidas. Este é o momento em que precisamos distribuir nossos atributos em tabelas diferentes para que a normalização seja possível.

A normalização é indispensável se queremos qualidade, mas, em alguns casos, ela pode impactar negativamente na performance das consultas e atualizações. Muitas vezes os projetistas de banco de dados precisam fazer alguma concessão nos níveis de normalização em razão da melhora do desempenho. (ELMASRI, 2018)

Outra característica importante de um projeto de banco de dados relacional é mecânica de indexação entre as diversas tabelas responsáveis pela persistência dos objetos. Quanto mais tabelas envolvidas, mais ligações entre elas serão necessárias e o tempo para consultas e atualizações serão proporcionalmente maiores. Para cada *link* entre tabelas, o SGBD (Sistema Gerenciador de Banco de Dados) cria arquivos de índices que consomem tempo para consulta e atualização quando necessário. Os SGBD's de mercado lidam muito bem com esses desafios e, nos casos mais comuns, isso não chega a prejudicar visivelmente o resultado. (SOMMERVILLE, 2019)

2.6. As classes do DAO (Data Access Object)

DAO é a camada do sistema que busca abstrair todo o acesso ao banco de dados separadamente da lógica de negócio da aplicação, e é aqui onde o projetista cria e mantém as classes capazes de recriarem os objetos à medida que for necessário. Normalmente, em sistemas com persistência em bancos relacionais, não se costuma manter listas residentes em memória e, em seu lugar, trabalhamos instâncias individuais dos objetos sob demanda. Existem disponíveis no mercado alguns *frameworks* de persistência como *Hibernate* e *TopLink* por exemplo, que são muito úteis e podem ajudar bastante neste trabalho. É possível garantir a integridade dos objetos porque assim que se faz qualquer atualização no objeto, é feita imediatamente a atualização das tabelas responsáveis pela sua persistência. Na próxima vez que for requisitado, o objeto é remontado a partir das tabelas atualizadas em sua última requisição. Se a modelagem do banco estiver correta, o SGBD possui mecanismos que conseguem garantir toda a integridade dos objetos. (ELMASRI, 2018)

2.7. Projetando um sistema OO para persistência por serialização

Nada precisa ser feito com relação a preparação de tabelas e/ou modelagem do banco de dados e, também não vamos precisar de um SGBD nem tão pouco de *frameworks* de persistência. Ao invés disso, o projetista terá que cuidar para que todos os registros (objetos) permaneçam o tempo todo residentes em memória e, sempre que ocorrer qualquer alteração, em qualquer um dos registros, os dados envolvidos deverão ser serializados e persistidos imediatamente. (PREISSLER, 2012)

2.8. Sistemas híbridos (SGBD/Serialização)

Copyright © 2023 ISESC

Em alguns casos podemos serializar objetos de montagem complexa ou demorada, objetos normalmente não identidade ou que representem uma "fotografia" de um determinado momento do negócio, como um relatório técnico por exemplo. Este tipo de objeto se presta bem para persistência via serialização e pode ser facilmente indexado. Com esta estratégia, podemos deixar o banco de dados mais leve e a recuperação dos objetos persistidos feitos em alta velocidade, passando a sensação para o usuário que os documentos são obtidos instantaneamente. Este tipo de objeto normalmente não sofre manutenção alguma, são apenas armazenados para eventuais consultas e por isso não são críticos com relação a problemas de integridade.

2.8.1. *Caching*

Caching é uma técnica de armazenamento de dados em memória usada para permitir o acesso mais rápido aos dados do que remontar o objeto utilizando diretamente a base de dados. Com os objetos pré-carregados, ganha-se tempo minimizando o acesso ao banco de dados evitando abertura de conexões, problemas de bloqueio de acesso etc. O uso de *caching* pode trazer muitos benefícios, mas precisa ser cuidadoso no aspecto de que não pode haver diferenças entre os atributos dos objetos em memória e os atributos deste objeto na base de dados. Quando se trabalha com *caching*, outras aplicações não podem fazer atualizações na base de dados, a menos que essas atualizações sejam previstas e tratadas adequadamente. Em sistemas híbridos, é muito comum que os dados em cache sejam serializados como forma de agilizar a carga de listas de uso frequente (SILVA, 2017)

3. Conclusões

Ambas as tecnologias oferecem garantias no que diz respeito a integridade dos dados desde que ambas observem os princípios básicos, técnicas e boas práticas recomendadas para cada caso. Todavia o trabalho em alto nível na operação com banco de dados exige domínio de técnicas sofisticadas que exigem treinamento específico e alguma experiência por parte da equipe de desenvolvimento. Já a serialização é um processo muito mais simples e que utiliza de artifícios intuitivos, acessíveis mesmo para os profissionais iniciantes. Basicamente o programador precisa manter listas em memória, serializá-las sempre que alteradas e o resto a OO deve dar conta.

A seguir, itens a considerar na escolha de tecnologia de persistência em sistemas OO.

a) Serialização:
   - Sistemas com relativamente poucos registros;
   - Sistemas que utilizam a técnica de *caching* de objetos; (híbrido)
   - Sistemas sem potencial para integração com outros sistemas;
   - Pequenos projetos com carência de recursos de *peopleware, software* e/ou *hardware*;
   - Troca de dados entre processos em sistemas distribuídos; (híbrido)
   - Quando se deseja "fotografar" o estado de um objeto; (híbrido)
   - Treinamento de desenvolvimento de sistemas quando os treinandos ainda não dominam modelagem de dados;
   - Agilidade no desenvolvimento da solução de *software* (menor tempo de desenvolvimento).

b) Banco de dados:
   - Sistemas com alto potencial de extensibilidade;
   - Sistemas com alto potencial de integração com outros sistemas legados;
   - Sistemas com alto potencial de distribuição.

Referências:

BERTOL. **Serialização de objetos em JAVA**. 2012. Disponível em: https://www.devmedia.com.br/serializacao-de-objetos-em-java/23413. Acesso em: 19 fev. 2023.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 7. ed. São Paulo, SP: Pearson, 2018. *E-book*. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 20 fev. 2023.

FÉLIX, R.(org.). **Programação orientada a objetos**. São Paulo, SP: Pearson, 2016. E-book. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 19 fev. 2023.

HENRIQUE, J**. POO: o que é programação orientada a objetos?** Disponível em: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos. Acesso em: 19 fev. 2023.

PREISSLER, L.E.; GEYER, C.F.R. - **Serialização em Java** Disponível em: http://www.inf.ufrgs.br/gppd/disc/cmp167/trabalhos/sem99-1/T1/luciano/final.htm Acesso em: 11 dez. 2022.

SILVA, C.A. **Entendendo serialização de objetos e o SERIALVERSIONUID. 2017.** Disponível em:

Copyright © 2023 ISESC

https://www.oracle.com/br/technical-resources/articles/java/serialversionuid.html Acesso em: 05 jan. 2023.

SOMMERVILLE, I**. Engenharia de software**. São Paulo: Pearson 2019

VERSOLATTO, F. **Sistemas orientados a objetos**: **conceitos e práticas**. Rio de Janeiro, RJ: Freitas Bastos, 2023. E-book. Disponível no formato EPUB em: https://plataforma.bvirtual.com.br. Acesso em: 15 fev. 2023.

Copyright © 2023 ISESC