Artigo Técnico

Persistência e jpa

Estudo comparativo entre as tecnologias JDBC, Hibernate com XML e JPA, contemplando Transação, Cache e Concorrência.

Persistência e jpa

Artigo Técnico

Tecnologias e api’s

Foram utilizadas:

- JDBC

-Hibernate 5 (XML Config)

- JPA 2.0

- H2 Database em todas as POC’s

Rodrigo Martinez da Rocha – RM48236

27SCJ

Professor Emílio Celso de Souza

Sumário

[1. Introdução 2](#_Toc456636655)

[2. Java JDBC 3](#_Toc456636656)

[2.1 Transação 3](#_Toc456636657)

[2.2 Cache 4](#_Toc456636658)

[2.3 Concorrência 5](#_Toc456636659)

[3. Hibernate com XML 7](#_Toc456636660)

[3.1 Transação 7](#_Toc456636661)

[3.2 Cache 8](#_Toc456636662)

[3.3 Concorrência 8](#_Toc456636663)

[4. JPA 9](#_Toc456636664)

[4.1 Transação 9](#_Toc456636665)

[4.2 Cache 9](#_Toc456636666)

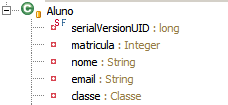
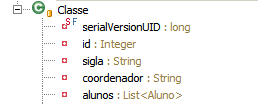
[4.3 Concorrência 10](#_Toc456636667)

# Introdução

Neste artigo iremos comparar e apresentar implementações relacionadas a camada de persistência Java, no que se diz respeito a: Controles Transacionais, Cache e Concorrência.

Em todos os casos optamos por utilizar o database H2, por sua praticidade e reprodutibilidade sem a necessidade de instalações adicionais de softwares vendor-in ou aplicações server-based.

As entidades utilizadas para este estudo foram:



Todas as inserções e busca tratadas no mesmo banco de dados, contendo a mesma estrutura.

Todos os códigos estão disponíveis e abertos em:

<https://github.com/rodrigogrohl/fiap-rrocha/tree/master/Persistencia-JPA/artigo-final/poc-source>

# Java JDBC

Os códigos aqui demonstrados estão implementados no projeto ‘jdbc-module’.

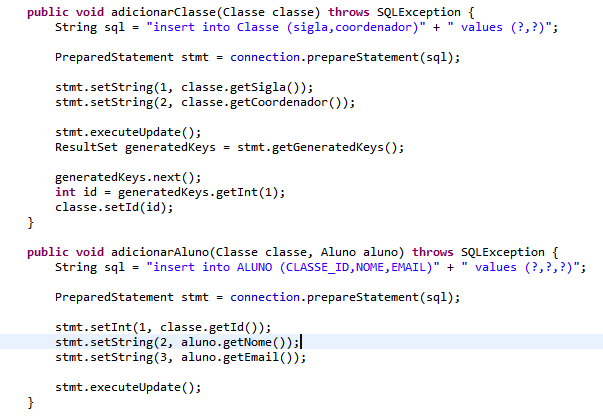
## Transação

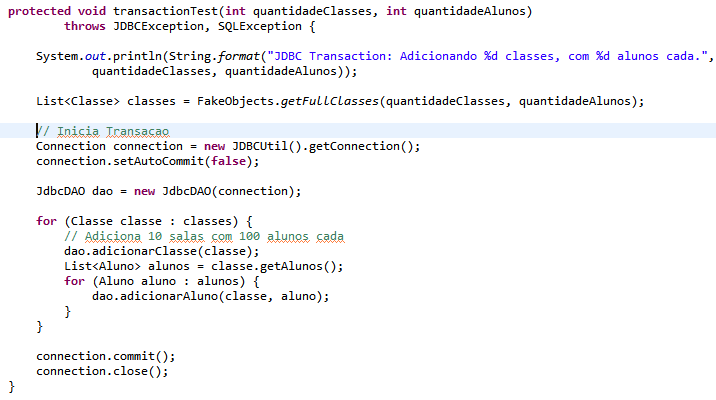
Vamos tratar o primeiro ponto de suma importância: Por padrão o JDBC cria suas conexões com a feature de auto-commit ativada, ou seja, para controlarmos o processo transacional, devemos, primeiramente, desativar este recurso:

.setAutoCommit(false);

Com isso teremos autonomia e controle transacional, obtendo a segurança necessária, principalmente em casos de rollback.

Nosso teste padrão será a inserção de 10 Classes com 100 Alunos cada, um total de 1000 entidades persistidas:





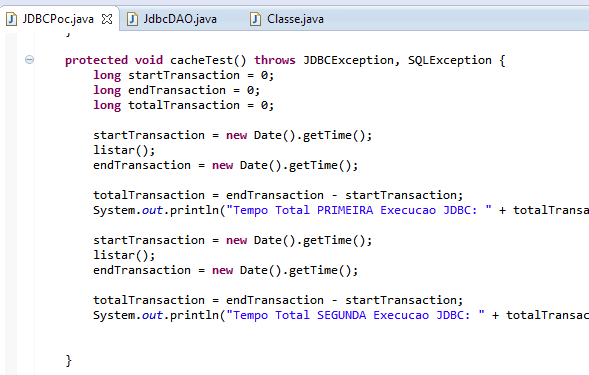
Resultando em:



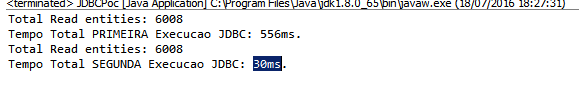
## Cache

O cache via JDBC pode ser implementado de maneira implícita ou explícita, via Statement Cache, responsável por manter as entidades em memória após a chamada do método ‘close()’.

Todos os códigos estão disponíveis na classe JDBCPoc:



Execução e resultado via console:

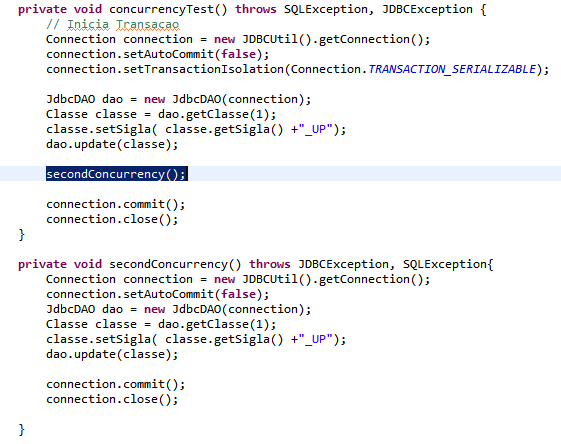


## Concorrência

Por definição o JDBC pode controlar o acesso concorrente a informação persistida através da configuração abaixo:

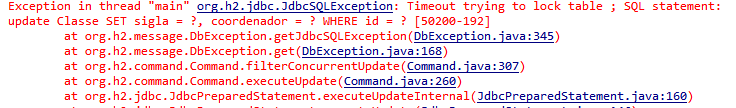


Nesta POC forçamos duas conexões paralelas o requerimento de update ao mesmo registro:



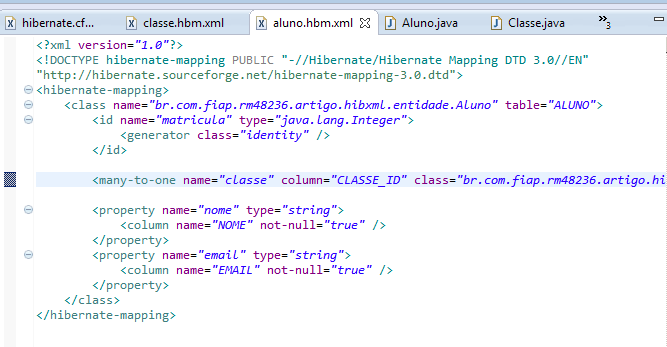
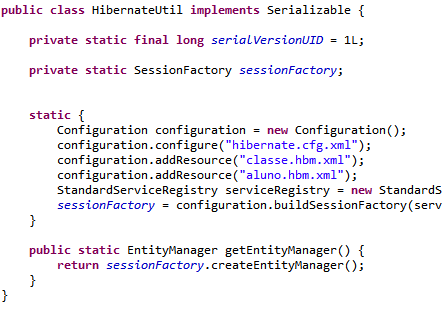
O código está disponível na classe JDBCPoc.java, dentro do método concurrencyTest().

Com isso temos a garantia de lock:



# Hibernate com XML

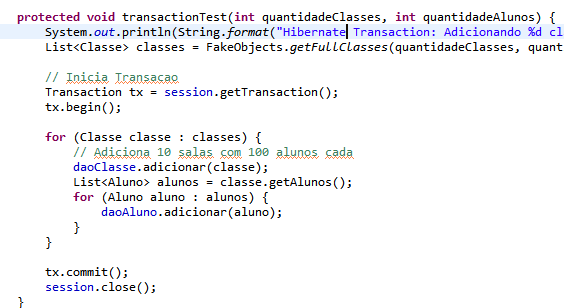
Utilizando o Hibernate, foi implementada a configuração global (hibernate.cfg.xml) e suas respectivas entidades (aluno.hbm.xml e classe.hbm.xml)



## Transação

O Controle de transações pode ser realizado em ambiente não gerenciado ou via JTA.

As implementações de exemplo e POC estão disponíveis em ‘JPAPoc().cacheTest()’



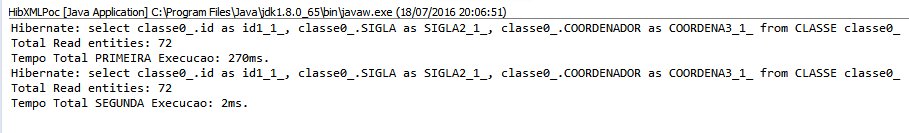
O bloco acima descreve uma transação que persiste 100 entidades em controle centralizado.

## Cache

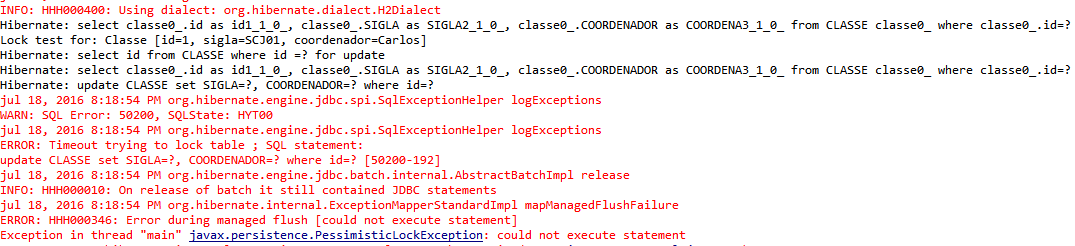
Como implementação de referência, o cache pode ser gerenciado a nível de entidades, e regiões específicas podem ser configuradas, otimizando entidades ou até mesmo priorizando queries com maior utilização e recorrência.

Em segundo nível, o hibernate suporta as seguintes implementações de cache: hastable (para desenvolvimento apenas), EHCache, OSCache, SwarmCache e JBossCache.

Os códigos de POC estão disponíveis em **JPAPoc().cacheTest()**



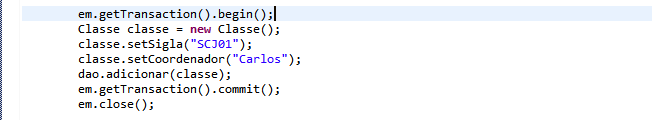
## Concorrência



# JPA

## Transação

As transações em contexto JPA, podem ser gerenciadas manualmente ou por um container, caso exista. Abaixo, temos um exemplo de recurso gerenciado de forma manual, e está disponível no projeto ‘jpa-module’:



Desta forma, temos toda a gestão e lógica da utilização do recurso. Fator que infere pontos positivos (como controle granular das operações) e pontos negativos (mau uso de recursos e pontos de falha nas implementações);

No teste de desempenho, obtivemos o seguinte resultado:



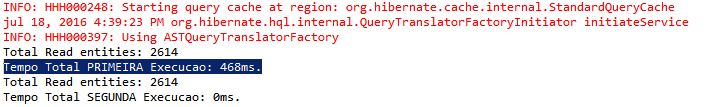
Operação disponível através do método (br.com.fiap.rm48236.artigo.jpa.Transacao.java): 

## Cache

No JPA temos disponível de forma nativa os seguintes tipos de cache:

**Cache de Primeiro Nível**: é nossa própria instância do EntityManager;  
 **Cache de Segundo Nível**: é configurado via API, onde é possível deixar em memória as linhas mais utilizadas dentro de um contexto, definindo apenas um provedor de cache (EhCache, OsCache, JBoss Cache) e um método de invalidação, que, normalmente é dado por atualização do registro no banco e/ou tempo ´pré-definido.  
 **Query Cache**: são utilizadas dentro do contexto do Cache de Segundo Nível, porém sob a ótima de uma query completa.

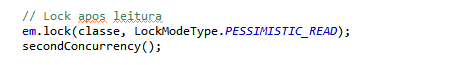
No teste de desempenho, disponível no método cacheTest() em Transacao.java realizamos duas buscas, a primeira vai ao banco, carrega o cache e retorna a lista solicitada, a mesma operação é repetida com a query em cache de segundo nível:



## Concorrência

Por especificação a API JPA utiliza a técnica de Lock para gerenciar concorrências em databases transacionais. Quando duas transações, de forma paralela tentam acessar a mesma informação, o lock é utilizado para garantir a consistência dos dados.

A implementação é suportada de forma nativa pela API, no método concurrencyTest() temos uma implementação de Lock, uma primeira transação lê o regsitro e seta o lock:



Durante a transação, uma segunda tenta atualizar o mesmo registro:



Uma exceção é disparada, garantido o lock:



A estratégia do lock é definida conforme necessidade, e deve ser implementada com muita cautela.

Referências:

Os 7 hábitos dos desenvolvedores Hibernate e JPA altamente eficazes  
<http://blog.caelum.com.br/os-7-habitos-dos-desenvolvedores-hibernate-e-jpa-altamente-eficazes/>

Locking and Concurrency in Java Persistence 2.0  
<https://blogs.oracle.com/enterprisetechtips/entry/locking_and_concurrency_in_java>

EhCache  
<http://www.ehcache.org/>

Statement and Result Set Caching  
<https://docs.oracle.com/database/121/JJDBC/stmtcach.htm#JJDBC28653>

JBoss Hibernate  
<https://docs.jboss.org/hibernate/orm/5.0/manual/en-US/html/ch03.html>

Todos os códigos estão disponíveis e abertos em:

<https://github.com/rodrigogrohl/fiap-rrocha/tree/master/Persistencia-JPA/artigo-final/poc-source>

Rodrigo Martinez da Rocha – RM48236

27SCJ

Professor Emílio Celso de Souza