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

[Digite o título da barra lateral]

[Digite o conteúdo da barra lateral. A barra lateral é um suplemento autônomo do documento principal. Ela está alinhada à esquerda/direita ou na parte superior/inferior da página. Use a guia Ferramentas de Desenho para alterar a formatação da caixa de texto da barra lateral.]

Sumário

[1. Java JDBC 3](#_Toc456597859)

[1.1 Transação 3](#_Toc456597860)

[1.2 Cache 3](#_Toc456597861)

[1.3 Concorrência 3](#_Toc456597862)

[2. Hibernate com XML 4](#_Toc456597863)

[2.1 Transação 4](#_Toc456597864)

[2.2 Cache 4](#_Toc456597865)

[2.3 Concorrência 4](#_Toc456597866)

[3. JPA 5](#_Toc456597867)

[3.1 Transação 5](#_Toc456597868)

[3.2 Cache 5](#_Toc456597869)

[3.3 Concorrência 5](#_Toc456597870)

# Java JDBC

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

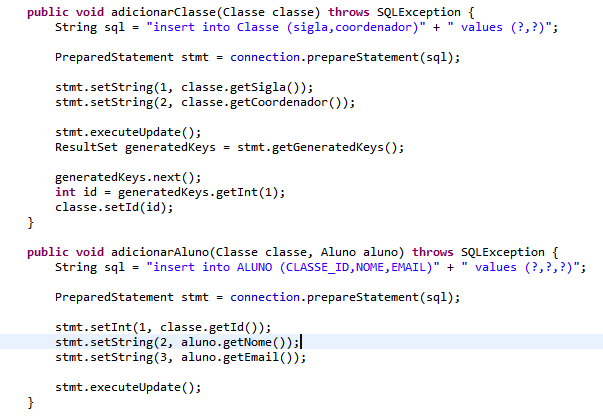
## Transação

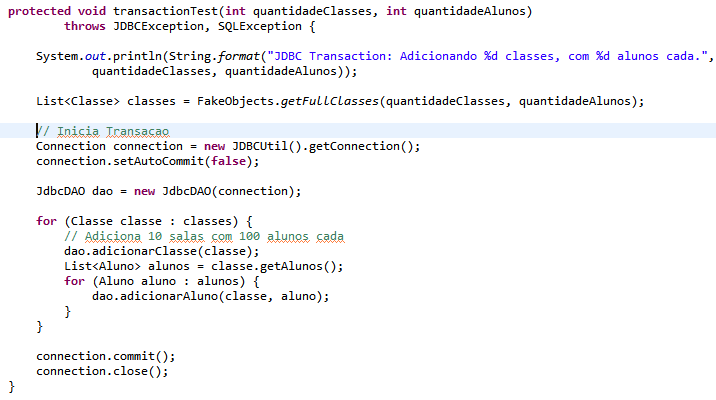
Vamos tratar o primeiro ponto de suma importância: Por padrão o JDBC cria suas conexões com a feature de auto-commit ativada, ou seja, para controlarmos o processo transacional, devemos, primeiramente, desativar este recurso:

.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:





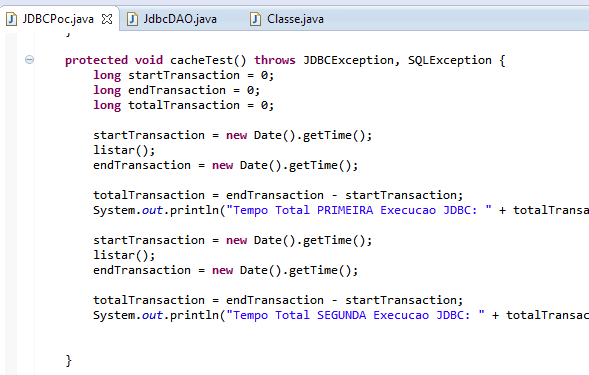
Resultando em:



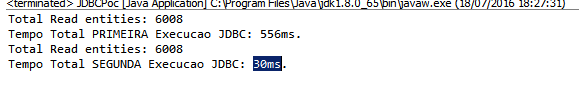
## Cache

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:

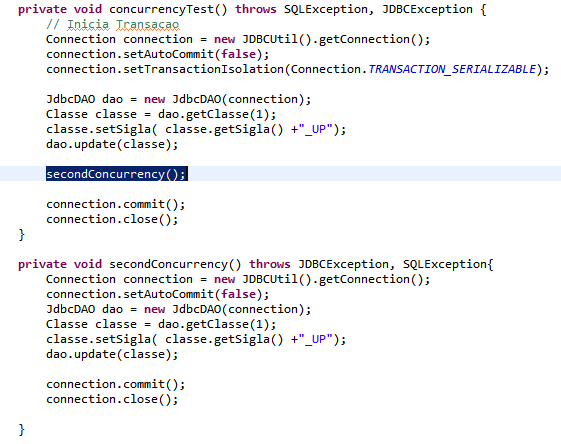


## Concorrência

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

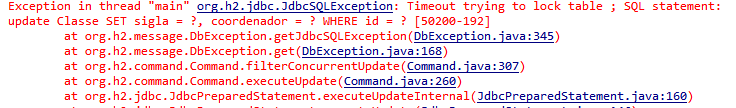


Nesta POC forçamos duas conexões paralelas o requerimento de update ao mesmo registro:



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

Com isso temos a garantia de lock:



# Hibernate com XML

## Transação

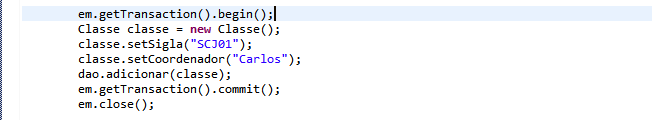
## Cache

## 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:



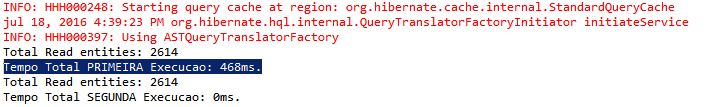
Operação disponível através do método (br.com.fiap.rm48236.artigo.jpa.Transacao.java): 

## 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.

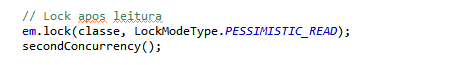
No teste de desempenho, disponível no método cacheTest() em Transacao.java realizamos duas buscas, a primeira vai ao banco, carrega o cache e retorna a lista solicitada, a mesma operação é repetida com a query em cache de segundo nível:



## 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>