Oracle Blog

Carol McDonald

Java Stammtisch

« JSF 2.0, JPA, GlassF... | Main | Java EE 6 Pet Catalo... »

JPA 2.0 Concurrency and locking

By caroljmcdonald on Jul 30, 2009

Optimistic Concurrency

Optimistic locking lets concurrent transactions process simultaneously, but detects and prevent collisions, this works best for applications where most concurrent transactions do not conflict. JPA Optimistic locking allows anyone to read and update an entity, however a version check is made upon commit and an exception is thrown if the version was updated in the database since the entity was read. In JPA for Optimistic locking you annotate an attribute with @Version as shown below:

```
public class Employee {
    @ID int id;
@Version int version;
```

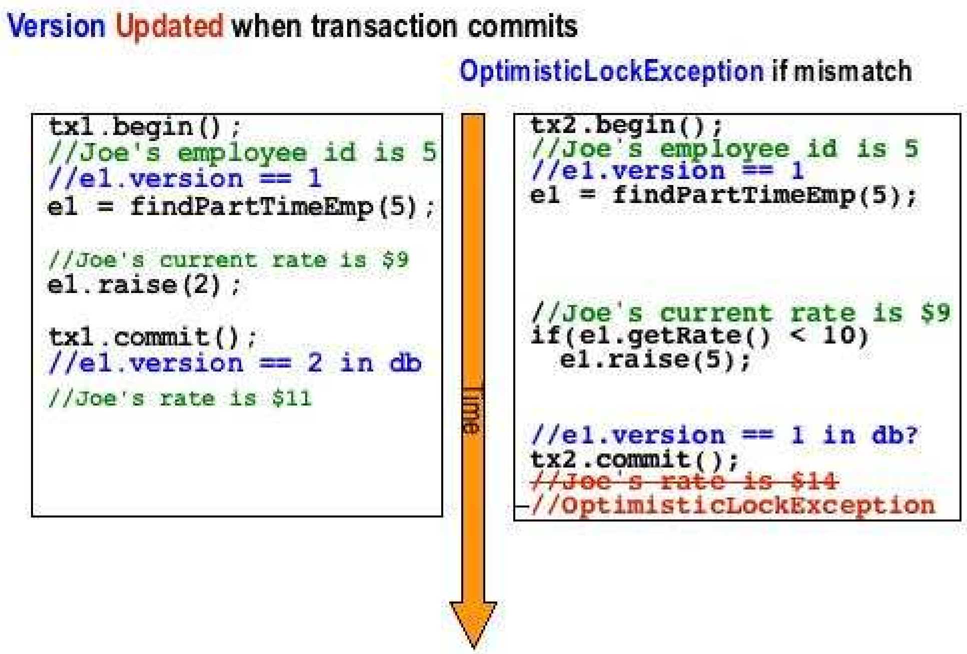
The Version attribute will be incremented with a successful commit. The Version attribute can be an int, short, long, or timestamp. This results in SQL like the following:

```
"UPDATE Employee SET ..., version = version + 1
     WHERE id = ? AND version = readVersion"
```

The advantages of optimistic locking are that no database locks are held which can give better scalability. The disadvantages are that the user or application must refresh and retry failed updates.

Optimistic Locking Example

In the optimistic locking example below, 2 concurrent transactions are updating employee **e1**. The transaction on the left commits first causing the **e1** version attribute to be incremented with the update. The transaction on the right throws an OptimisticLockException because the e1 version attribute is higher than when **e1** was read, causing the transaction to roll back.


Version Updated when transaction commits
OptimisticLockException if mismatch
Time

```
tx1.begin();
//Joe's employee id is 5
//e1.version == 1
e1 = findPartTimeEmp(5);

//Joe's current rate is $9
e1.raise(2);

tx1.commit();
//e1.version == 2 in db
//Joe's rate is $11
```

```
tx2.begin();
//Joe's employee id is 5
//e1.version == 1
e1 = findPartTimeEmp(5);



//Joe's current rate is $9
if(e1.getRate() < 10)
  e1.raise(5);

//e1.version == 1 in db?
tx2.commit();
~~//Joe's rate is $14~~
//OptimisticLockException
```

Additional Locking with JPA Entity Locking APIs

With JPA it is possible to lock an entity, this allows you to control when, where and which kind of locking to use. JPA 1.0 only supported Optimistic read or Optimistic write locking. JPA 2.0 supports Optimistic and Pessimistic locking, this is layered on top of @Version checking described above.

JPA 2.0 LockMode values :

- OPTIMISTIC (JPA 1.0 READ):
  - perform a version check on locked Entity before commit, throw an OptimisticLockException if Entity version mismatch.
- OPTIMISTIC_FORCE_INCREMENT (JPA 1.0 WRITE)
  - perform a version check on locked Entity before commit, throw an OptimisticLockException if Entity version mismatch, force an increment to the version at the end of the transaction, even if the entity is not modified.

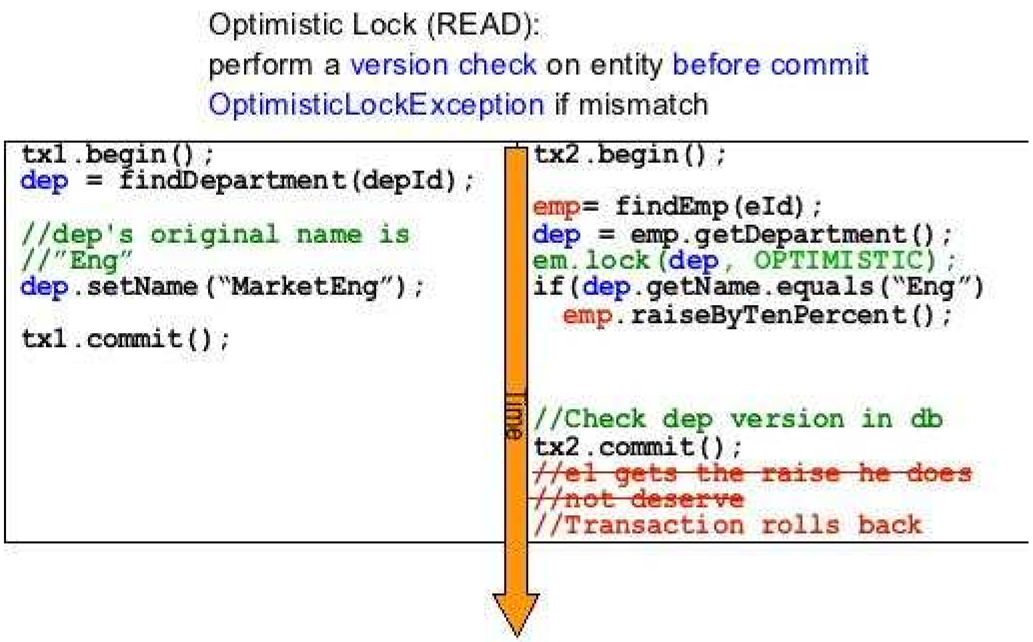
- PESSIMISTIC:
  - lock the database row when reading
- PESSIMISTIC_FORCE_INCREMENT
  - lock the database row when reading, force an increment to the version at the end of the transaction, even if the entity is not modified.

There are multiple APIs to specify locking an Entity:

- EntityManager methods: lock, find, refresh
- Query methods: setLockMode
- NamedQuery annotation: lockMode element

OPTIMISTIC (READ) LockMode Example

In the optimistic locking example below, transaction1 on the left updates the department name for **dep** , which causes **dep**'s version attribute to be incremented. Transaction2 on the right gives an **emp**loyee a raise if he's in the "Eng" department. Version checking on the employee attribute would not throw an exception in this example since it was the **dep** Version attribute that was updated in transaction1. In this example the employee change should not commit if the department was changed after reading, so an OPTIMISTIC lock is used : **`em.lock(dep, OPTIMISTIC)`**. This will cause a version check on the **dep** Entity before committing transaction2 which will throw an OptimisticLockException because the **dep** version attribute is higher than when **dep** was read, causing the transaction to roll back.



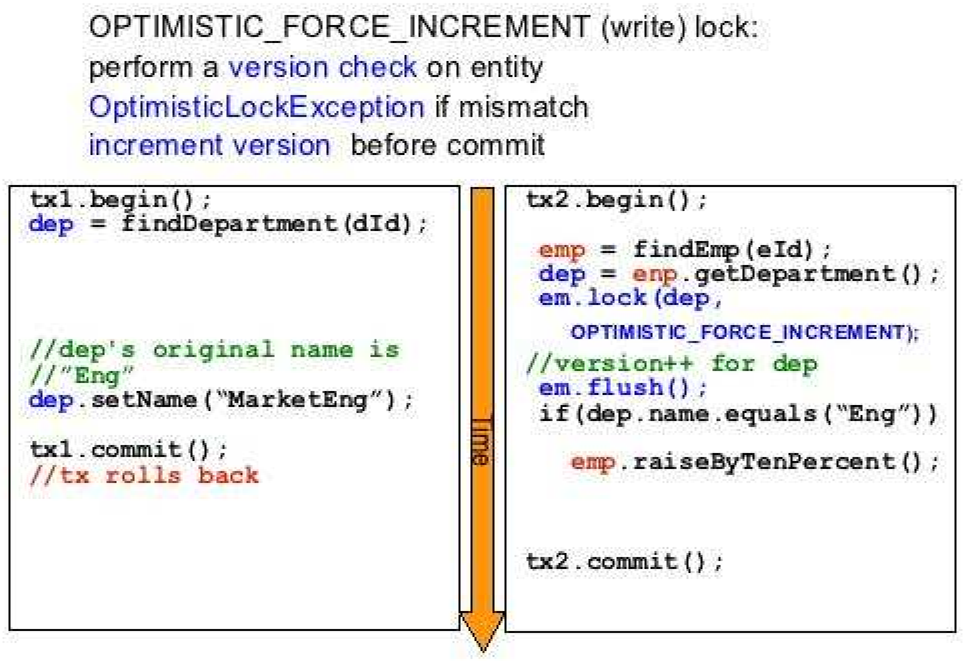
Optimistic Lock (READ):
perform a version check on entity before commit
OptimisticLockException if mismatch

```
tx1.begin();
dep = findDepartment(depId);

//dep's original name is
//"Eng"
dep.setName("MarketEng");

tx1.commit();
```

```
tx2.begin();

emp= findEmp(eId);
dep = emp.getDepartment();
em.lock(dep, OPTIMISTIC);
if(dep.getName.equals("Eng")
   emp.raiseByTenPercent();


//Check dep version in db
tx2.commit();
~~//e1 gets the raise he does~~
~~//not deserve~~
//Transaction rolls back
```

OPTIMISTIC_FORCE_INCREMENT (write) LockMode Example

In the OPTIMISTIC_FORCE_INCREMENT locking example below, transaction2 on the right wants to be sure that the dep name does not change during the transaction, so transaction2 locks the **dep** Entity **`em.lock(dep, OPTIMISTIC_FORCE_INCREMENT)`** and then calls em.flush() which causes **dep**'s version attribute to be incremented in the database. This will cause any parallel updates to **dep** to throw an OptimisticLockException and roll back. In transaction1 on the left at commit time when the **dep** version attribute is checked and found to be stale, an OptimisticLockException is thrown


OPTIMISTIC_FORCE_INCREMENT (write) lock:
perform a version check on entity
OptimisticLockException if mismatch
increment version before commit

```
tx1.begin();
dep = findDepartment(dId);



//dep's original name is
//"Eng"
dep.setName("MarketEng");

tx1.commit();
//tx rolls back
```

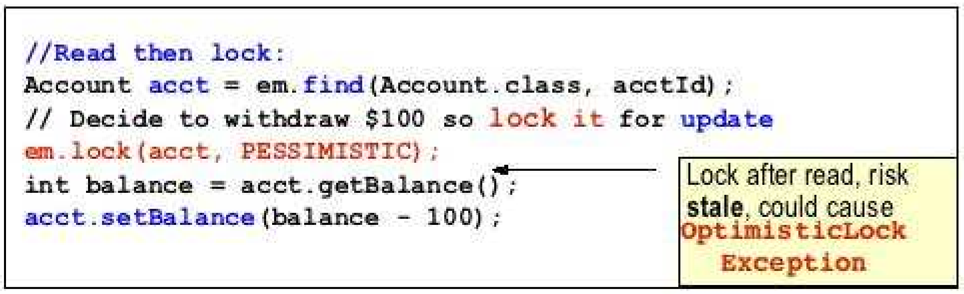
```
tx2.begin();

 emp = findEmp(eId);
 dep = emp.getDepartment();
 em.lock(dep,
    OPTIMISTIC_FORCE_INCREMENT);
//version++ for dep
 em.flush();
 if(dep.name.equals("Eng"))
    emp.raiseByTenPercent();


tx2.commit();
```

Pessimistic Concurrency

Pessimistic concurrency locks the database row when data is read, this is the equivalent of a (SELECT . . . FOR UPDATE [NOWAIT]) . Pessimistic locking ensures that transactions do not update the same entity at the same time, which can simplify application code, but it limits concurrent access to the data which can cause bad scalability and may cause deadlocks. Pessimistic locking is better for applications with a higher risk of contention among concurrent transactions.
The examples below show:

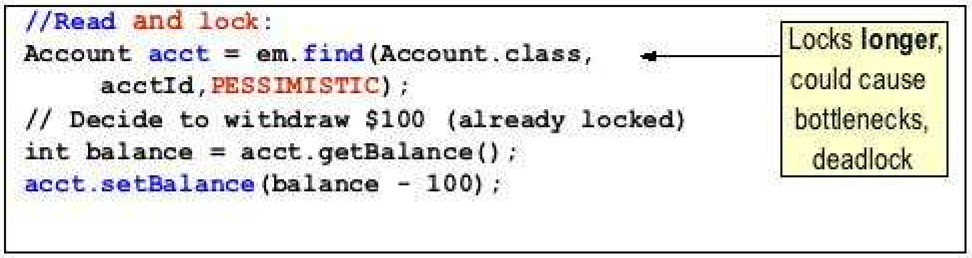
1. reading an entity and then locking it later
2. reading an entity with a lock
3. reading an entity, then later refreshing it with a lock

The Trade-offs are the longer you hold the lock the greater the risks of bad scalability and deadlocks. The later you lock the greater the risk of stale data, which can then cause an optimistic lock exception, if the entity was updated after reading but before locking.

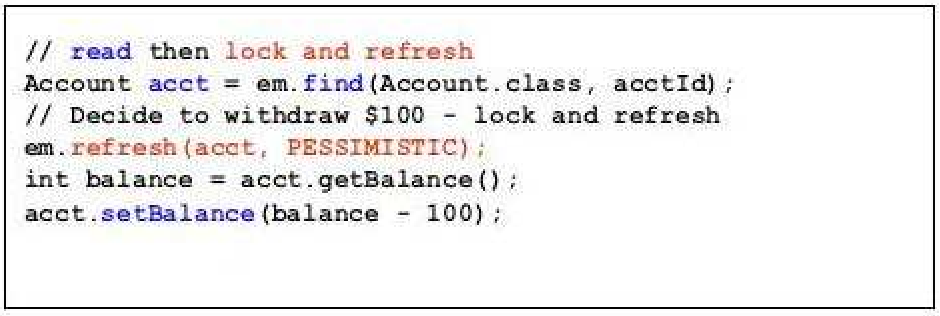


```
//Read then lock:
Account acct = em.find(Account.class, acctId);
// Decide to withdraw $100 so lock it for update
em.lock(acct, PESSIMISTIC);
int balance = acct.getBalance();
acct.setBalance(balance - 100);
```

Lock after read, risk **stale**, could cause OptimisticLock Exception



```
//Read and lock:
Account acct = em.find(Account.class,
      acctId,PESSIMISTIC);
// Decide to withdraw $100 (already locked)
int balance = acct.getBalance();
acct.setBalance(balance - 100);
```



```
// read then lock and refresh
Account acct = em.find(Account.class, acctId);
// Decide to withdraw $100 - lock and refresh
em.refresh(acct, PESSIMISTIC);
int balance = acct.getBalance();
acct.setBalance(balance - 100);
```

The right locking approach depends on your application:

- what is the risk of risk of contention among concurrent transactions?
- What are the requirements for scalability?
- What are the requirements for user re-trying on failure?

References and More Information:

Preventing Non-Repeatable Reads in JPA Using EclipseLink
Java Persistence API 2.0: What's New ?
What's New and Exciting in JPA 2.0
Beginning Java™ EE 6 Platform with GlassFish™ 3
Pro EJB 3: Java Persistence API (JPA 1.0)

Java Persistence API: Best Practices and Tips

Category: Sun

Tags: glassfish javaee jpa

Permanent link to this entry

« JSF 2.0, JPA, GlassF... | Main | Java EE 6 Pet Catalo... »
Comments:

It's nice to see, that pessimitic locking is available in JPA 2.0 now. In my experience a very solid approach for getting very scalable solutions is this:

- Use optimistic locking all the time.
- Addionally do pessimistic locking yourself in your application depending on your domain.

With this approach you do not get "blocks" or dead locks in the pessimistic locking part because you code the locking yourself. And you can choose the right granularity and domain objects for locking in your domain.

Still you have the optimistic locking feature because there will be situations you missed. The optimistic locking then prevents you from data corruption. If you are working in a message driven bean you can just try again, for other situations optimistic locking can be your safety net.

Posted by **Tobias Frech** on August 17, 2009 at 02:45 AM EDT #

Post a Comment:
Comments are closed for this entry.

About

caroljmcdonald

Search

Enter search term:

☑ Search only this blog

Recent Posts

- Working for Sun, Oh, the places I have been
- Wicket, JPA, GlassFish and Java Derby or MySQL
- Java Garbage Collection, Monitoring and Tuning
- Web Site Performance Tips and Tools
- The Top 10 Web Application security vulnerabilities
- OWASP Top 10 number 3: Malicious File Execution
- Top 10 web security vulnerabilities number 2: Injection Flaws
- The Top 10 Web Application security vulnerabilities starting with XSS
- Some Concurrency Tips
- JPA Performance, Don't Ignore the Database

Top Tags

- ajax
- bayeux
- collection
- comet
- concurrency
- dojo
- eclipselink
- economy
- ejb
- esapi
- esb
- garbage
- glassfish
- grails
- grizzly
- groovy
- hibernate
- java
- javaee
- javafx
- javascript
- jax-rs
- jax-ws
- jmaki
- jpa
- jruby
- jsf
- memory
- monitoring
- mysql
- netbeans
- owasp
- performance
- persistence
- rails
- rest
- seam
- security
- services
- spring
- tuning
- web
- wicket
- xss

Categories

- Personal
- Sun

Archives

« April 2012

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Today

Bookmarks

- blogs.sun.com
- java.com
- java.net
- opensolaris.org

Menu

- Blogs Home
- Weblog
- Login

Feeds

RSS

- All
- /Personal
- /Sun
- Comments

Atom

- All
- /Personal
- /Sun
- Comments