JUDCon

JBoss Users & Developers Conference

London:2011

Hibernate Puzzlers

Patrycja Wegrzynowicz
CTO of Yonita, Inc.

About Me

- 10+ years of professional experience as software developer, architect, and head of software R&D
- PhD in Computer Science
  - Patterns and anti-patterns, code analysis, language semantics, compiler design
- Speaker at JavaOne, Devoxx, OOPSLA, JavaZone, others
- CTO of Yonita, Inc.
  - Bridge the gap between the industry and the academia
  - Automated detection of software defects
  - Security, performance, concurrency, databases

Today

- **Five** hibernate-related puzzles
  - Hibernate, JPA, general database issues
  - Correctness, performance
- Short program with curious behavior
- Question to you (multiple choice)
- Mystery revealed
- How to fix it
- Lessons learned

Disclaimer:
I do think Hibernate is great!

#1: Less Is More

Who knows

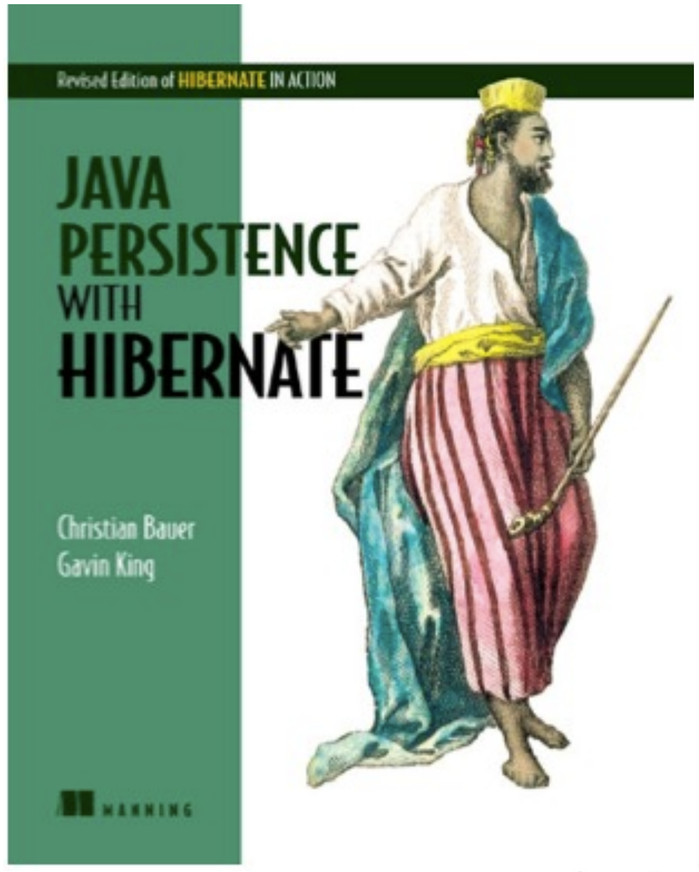
CaveatEmptor app?

Who has read

Java Persistence with Hibernate?

CaveatEmptor

- Demo application from ‘Java Persistence with Hibernate’
- Simple auction system



I'm Gonna Win This Auction!

```
public class PlaceBidAction implements Action {
   // invoked in a new transaction
   public String execute(Map even) {
     // ...
      Bid currentMinBid = itemDAO.getMinBid(itemId);
      Bid currentMaxBid = itemDAO.getMaxBid(itemId);

      // uses load with Lock.UPGRADE
      Item item = itemDAO.findById(itemId, true);
      Bid newBid = item.placeBid(userDAO.findById(userId, false),
                                 newAmount,
                                 currentMaxBid,
                                 currentMinBid);
     // ...
   }
}
```

What Is the Problem?

(a) None (code is correct)
(b) Transaction/lock mgmt
(c) Exception thrown
(d) None of the above

```
public class PlaceBidAction implements Action {
   // invoked in a new transaction
   public String execute(Map even) {
     // ...
      Bid currentMinBid = itemDAO.getMinBid(itemId);
      Bid currentMaxBid = itemDAO.getMaxBid(itemId);

      // uses load with Lock.UPGRADE
      Item item = itemDAO.findById(itemId, true);
      Bid newBid = item.placeBid(userDAO.findById(userId, false),
                                 newAmount,
                                 currentMaxBid,
                                 currentMinBid);
     // ...
   }
}
```

What Is the Problem?

(a) None (the code is correct)
(b) Transaction/Lock management
(c) Exception thrown
(d) None of the aboves

The bids can change between calling getMaxBid and locking the item.

Another Look

```
public class PlaceBidAction implements Action {
   // invoked in a new transaction
   public String execute(Map even) {
     // ...
      Bid currentMinBid = itemDAO.getMinBid(itemId);
      Bid currentMaxBid = itemDAO.getMaxBid(itemId);
      // at this moment, the bids can be changed by a different thread
      // uses load with Lock.UPGRADE
      Item item = itemDAO.findById(itemId, true);
      Bid newBid = item.placeBid(userDAO.findById(userId, false),
                                 newAmount,
                                 currentMaxBid,
                                 currentMinBid);
     // ...
   }
}
```

2 Users and Item with 2 Bids

Thread A (new amount 100)

(1)

```
// begin transaction
// item with 2 bids: 10,20
Bid curMax = dao.getMaxBid(id);
// curMax = 20
```

Thread B (new amount 30)

2 Users and Item with 2 Bids

Thread A (new amount 100)

(1)

```
// begin transaction
// item with 2 bids: 10,20
Bid curMax = dao.getMaxBid(id);
// curMax = 20
```

Thread B (new amount 30)

(2)

```
// begin transaction
// item with 2 bids: 10,20
Bid curMax = dao.getMaxBid(id);
// curMax = 20
```

2 Users and Item with 2 Bids

Thread A (new amount 100)

(1)

```
// begin transaction
// item with 2 bids: 10,20
Bid curMax = dao.getMaxBid(id);
// curMax = 20
```

(3)

```
// item with 2 bids: 10,20
Item item = dao.findById(id,true);
Bid newBid = item.placeBid(...,
            newAmount,
            curMax, ...);
// commit transaction
// item with 3 bids: 10,20,100
```

Successful bid: 100

Thread B (new amount 30)

(2)

```
// begin transaction
// item with 2 bids: 10,20
Bid curMax = dao.getMaxBid(id);
// curMax = 20
```

2 Users and Item with 2 Bids

Thread A (new amount 100)

(1)

```
// begin transaction
// item with 2 bids: 10,20
Bid curMax = dao.getMaxBid(id);
// curMax = 20
```

(3)

```
// item with 2 bids: 10,20
Item item = dao.findById(id,true);
Bid newBid = item.placeBid(...,
          newAmount,
          curMax, ...);
// commit transaction
// item with 3 bids: 10,20,100
```

Successful bid: 100

Thread B (new amount 30)

(2)

```
// begin transaction
// item with 2 bids: 10,20
Bid curMax = dao.getMaxBid(id);
// curMax = 20
```

(4)

```
// item with 3 bids: 10,20,100
      Item item = dao.findById(id,true);
Bid newBid = item.placeBid(...,
          newAmount,
          curMax, ...);
// commit transaction
// item with 4 bids: 10,20,100,30
```

Successful bid: 30

How To Fix It?

```
public class PlaceBidAction implements Action {
   // invoked in a new transaction
   public String execute(Map even) {
     // ...
      Bid currentMinBid = itemDAO.getMinBid(itemId);
      Bid currentMaxBid = itemDAO.getMaxBid(itemId);

      // uses load with Lock.UPGRADE
      Item item = itemDAO.findById(itemId, true);
      Bid newBid = item.placeBid(userDAO.findById(userId, false),
                                 newAmount,
                                 currentMaxBid,
                                 currentMinBid);
     // ...
   }
}
```

How To Fix It?

version and optimistic locking

```
ion implements Action {
ransaction
(Map even) {
  // ...
   Bid currentMinBid = itemDAO.getMinBid(itemId);
   Bid currentMaxBid = itemDAO.getMaxBid(itemId);

   // uses load with Lock.UPGRADE
   Item item = itemDAO.findById(itemId, true);
   Bid newBid = item.placeBid(userDAO.findById(userId, false),
                              newAmount,
                              currentMaxBid,
                              currentMinBid);
  // ...
 }
}
```

How To Fix It?

version and optimistic locking

REPEATABLE_READS

```
ion implements Action {
ransaction
(Map even) {
  // ...
   Bid currentMinBid = itemDAO.getMinBid(itemId);
   Bid currentMaxBid = itemDAO.getMaxBid(itemId);

   // uses load with Lock.UPGRADE
   Item item = itemDAO.findById(itemId, true);
   Bid newBid = item.placeBid(userDAO.findById(userId, false),
                              newAmount,
                              currentMaxBid,
                              currentMinBid);
  // ...
 }
}
```

How To Fix It?

version and optimistic locking

REPEATABLE_READS

SERIALIZABLE

```
ion implements Action {
ransaction
(Map even) {
    // ...
     Bid currentMinBid = itemDAO.getMinBid(itemId);
     Bid currentMaxBid = itemDAO.getMaxBid(itemId);

     // uses load with Lock.UPGRADE
     Item item = itemDAO.findById(itemId, true);
     Bid newBid = item.placeBid(userDAO.findById(userId, false),
                                newAmount,
                                currentMaxBid,
                                currentMinBid);
    // ...
  }
}
```

How To Fix It?

version and optimistic locking

REPEATABLE_READS

SERIALIZABLE

```
ion implements Action {
ransaction
(Map even) {
    // ...
    Bid currentMinBid = itemDAO.getMinBid(itemId);
    Bid currentMaxBid = itemDAO.getMaxBid(itemId);

    // uses load with Lock.UPGRADE
    Item item = itemDAO.findById(itemId, true);
    Bid newBid = item.placeBid(userDAO.findById(userId, false),
                               newAmount,
                               currentMaxBid,
                               currentMinBid);

    // ...
  }
}
```

move lock before getMinBid/getMaxBid

How To Fix It?

version and optimistic locking

REPEATABLE_READS

SERIALIZABLE

```
ion implements Action {
ransaction
(Map even) {
    // ...
    Bid currentMinBid = itemDAO.getMinBid(itemId);
    Bid currentMaxBid = itemDAO.getMaxBid(itemId);

    // uses load with Lock.UPGRADE
    Item item = itemDAO.findById(itemId, true);
    Bid newBid = item.placeBid(userDAO.findById(userId, false),
                               newAmount,
                               currentMaxBid,
                               currentMinBid);

    // ...
  }
}
```

move lock before getMinBid/getMaxBid

different object model

How To Fix It?

version and optimistic locking

NO

```
ion implements Action {
ransaction
(Map even) {
    // ...
    Bid currentMinBid = itemDAO.getMinBid(itemId);
    Bid currentMaxBid = itemDAO.getMaxBid(itemId);

    // uses load with Lock.UPGRADE
    Item item = itemDAO.findById(itemId, true);
    Bid newBid = item.placeBid(userDAO.findById(userId, false),
                               newAmount,
                               currentMaxBid,
                               currentMinBid);
    // ...
  }
}
```

REPEATABLE_READS

SERIALIZABLE

move lock before getMinBid/getMaxBid

different object model

How To Fix It?

version and optimistic locking — NO

REPEATABLE_READS — NO

SERIALIZABLE

```
ion implements Action {
ransaction
(Map even) {
    // ...
    Bid currentMinBid = itemDAO.getMinBid(itemId);
    Bid currentMaxBid = itemDAO.getMaxBid(itemId);

    // uses load with Lock.UPGRADE
    Item item = itemDAO.findById(itemId, true);
    Bid newBid = item.placeBid(userDAO.findById(userId, false),
                               newAmount,
                               currentMaxBid,
                               currentMinBid);

    // ...
  }
}
```

move lock before getMinBid/getMaxBid

different object model

How To Fix It?

version and optimistic locking — NO

REPEATABLE_READS — NO

SERIALIZABLE — YES

```
ion implements Action {
ransaction
(Map even) {
  // ...
   Bid currentMinBid = itemDAO.getMinBid(itemId);
   Bid currentMaxBid = itemDAO.getMaxBid(itemId);

   // uses load with Lock.UPGRADE
   Item item = itemDAO.findById(itemId, true);
   Bid newBid = item.placeBid(userDAO.findById(userId, false),
                              newAmount,
                              currentMaxBid,
                              currentMinBid);

  // ...
 }
}
```

move lock before getMinBid/getMaxBid

different object model

How To Fix It?

version and optimistic locking — NO

REPEATABLE_READS — NO

SERIALIZABLE — YES

```
ion implements Action {
ransaction
(Map even) {
    // ...
    Bid currentMinBid = itemDAO.getMinBid(itemId);
    Bid currentMaxBid = itemDAO.getMaxBid(itemId);

    // uses load with Lock.UPGRADE
    Item item = itemDAO.findById(itemId, true);
    Bid newBid = item.placeBid(userDAO.findById(userId, false),
                               newAmount,
                               currentMaxBid,
                               currentMinBid);

    // ...
  }
}
```

move lock before getMinBid/getMaxBid — YES

different object model

How To Fix It?

version and optimistic locking — NO

REPEATABLE_READS — NO

SERIALIZABLE — YES

```
ion implements Action {
ransaction
(Map even) {
    // ...
    Bid currentMinBid = itemDAO.getMinBid(itemId);
    Bid currentMaxBid = itemDAO.getMaxBid(itemId);

    // uses load with Lock.UPGRADE
    Item item = itemDAO.findById(itemId, true);
    Bid newBid = item.placeBid(userDAO.findById(userId, false),
                               newAmount,
                               currentMaxBid,
                               currentMinBid);

    // ...
  }
}
```

move lock before getMinBid/getMaxBid — YES

different object model — YES

The Simplest Solution

```
public class PlaceBidAction implements Action {
   // invoked in a new transaction
   public String execute(Map even) {
     // ...
      // uses load with Lock.UPGRADE
      Item item = itemDAO.findById(itemId, true);

      Bid currentMinBid = itemDAO.getMinBid(itemId);
      Bid currentMaxBid = itemDAO.getMaxBid(itemId);

      Bid newBid = item.placeBid(userDAO.findById(userId, false),
                                 newAmount,
                                 currentMaxBid,
                                 currentMinBid);
     // ...
   }
}
```

Different Object Model

- Calculate maximum in Java
  - Requires loading of all bids
- Keep track of the winning bid
  - Manual tracking (encapsulation needed)
  - Db denormalized but fast
- Utilize custom mappings with custom queries, laziness, order or where clause
  - Db normalized, as fast as present code

Lessons Learned

- Nobody's perfect ;-)
- ACID is only a theory
  - Simple transaction interceptor often is not enough
  - Query order does matter
  - Learn more about locks and transaction isolation levels
- Premature optimization is the root of all evil

#2: Volatile Warehouse

Volatile Warehouse

```
@Entity
public class Warehouse {
    private Long id;
    private int maxCapacity;
    private int actualUtilization;

    @Id @GeneratedValue
    public Long getId() {...}
    protected void setId() {...}

    public int getMaxCapacity() {...}
    protected void setCapacity(int c) {...}

    public void setActualUtilization(int u) {
        if (u > capacity)
            throw new IllegalArgumentException("utilization exceeds capacity");
        this.utilization = u;
    }
}

// new EntityManager and new transaction
Warehouse warehouse = new Warehouse(20, 10);
em.persist(warehouse);

// new EntityManager and new transaction
Warehouse found = em.find(Warehouse.class, warehouse.getId());
System.out.println(found.getMaxCapacity() + " " + found.getActualUtilization());
```

What Does It Print?

(a) 20 10
(b) 10 20
(c)Throws IllegalArgumentEx.
(d) Throws PersistentException
(e) None of the above

```
@Entity
public class Warehouse {
    private Long id;
    private int maxCapacity;
    private int actualUtilization;

    @Id @GeneratedValue
    public Long getId() {...}
    protected void setId() {...}

    public int getMaxCapacity() {...}
    protected void setCapacity(int c) {...}

    public void setActualUtilization(int u) {
        if (u > capacity)
            throw new IllegalArgumentException("utilization exceeds capacity");
        this.utilization = u;
    }
}

// new EntityManager and new transaction
Warehouse warehouse = new Warehouse(20, 10);
em.persist(warehouse);

// new EntityManager and new transaction
Warehouse found = em.find(Warehouse.class, warehouse.getId());
System.out.println(found.getMaxCapacity() + " " + found.getActualUtilization());
```

What Does It Print?

(a) 20 10
(b) 10 20
(c) Throws IllegalArgumentException
(d) Throws PersistentException
(e) None of the above

Hibernate sets the properties in the alphabetical order and **maxCapacity** is not initialized yet while setting **actualUtilization**.

IllegalArgumentException is wrapped by PersistentException.

Another Look

```
@Entity
public class Warehouse {
    private Long id;
    private int maxCapacity;
    private int actualUtilization;

    @Id @GeneratedValue
    public Long getId() {...}
    protected void setId() {...}

    public int getMaxCapacity() {...}
    protected void setCapacity(int c) {...} // called 2nd

    public void setActualUtilization(int u) { // called 1st
        if (u > capacity)
            throw new IllegalArgumentException("utilization exceeds capacity");
        this.utilization = u;
    }
}

// new EntityManager and new transaction
Warehouse warehouse = new Warehouse(20, 10);
em.persist(warehouse);

// new EntityManager and new transaction
Warehouse found = em.find(Warehouse.class, warehouse.getId());
System.out.println(found.getMaxCapacity() + " " + found.getActualUtilization());
```

We Can Make It Working

```
@Entity
public class Warehouse {
    private Long id;
    private int maxCapacity;
    private int utilization;

    @Id @GeneratedValue
    public Long getId() {...}
    protected void setId() {...}

    public int getMaxCapacity() {...}
    protected void setCapacity(int c) {...}

    public void setUtilization(int u) {
        if (u > capacity)
            throw new IllegalArgumentException("utilization exceeds capacity");
        this.utilization = u;
    }
}

// new EntityManager and new transaction
Warehouse warehouse = new Warehouse(20, 10);
em.persist(warehouse);

// new EntityManager and new transaction
Warehouse found = em.find(Warehouse.class, warehouse.getId());
System.out.println(found.getMaxCapacity() + " " + found.getActualUtilization());
```

We Can Make It Working

```
@Entity
public class Warehouse {
    private Long id;
    private int maxCapacity;
    private int utilization;

    @Id @GeneratedValue
    public Long getId() {...}
    protected void setId() {...}

    public int getMaxCapacity() {...}
    protected void setCapacity(int c) {...}

    public void setUtilization(int u) {
        if (u > capacity)
            throw new IllegalArgumentException("utilization exceeds capacity");
        this.utilization = u;
    }
}

// new EntityManager and new transaction
Warehouse warehouse = new Warehouse(20, 10);
em.persist(warehouse);

// new EntityManager and new transaction
Warehouse found = em.find(Warehouse.class, warehouse.getId());
System.out.println(found.getMaxCapacity() + " " + found.getActualUtilization());
```

It is NOT a good fix!
(think about maintanability)

Good Fix

```
@Entity
public class Warehouse {
    @Id @GeneratedValue
    private Long id;
    private int maxCapacity;
    private int actualUtilization;

    public Long getId() {...}
    protected void setId() {...}

    public int getMaxCapacity() {...}
    protected void setCapacity(int c) {...}

    public void setActualUtilization(int u) {
        if (u > capacity)
            throw new IllegalArgumentException("utilization exceeds capacity");
        this.utilization = u;
    }
}

// new EntityManager and new transaction
Warehouse warehouse = new Warehouse(20, 10);
em.persist(warehouse);

// new EntityManager and new transaction
Warehouse found = em.find(Warehouse.class, warehouse.getId());
System.out.println(found.getMaxCapacity() + " " + found.getActualUtilization());
```

Lessons Learned

- Property access mapping and rich domain model do not get along very well
  - JPA unspecified behavior
- Use field access mappings
  - Fields initialized without calling setters

#3: Heads of Hydra

Heads of Hydra

```
@Entity
public class Hydra {
   private Long id;
   private List<Head> heads = new ArrayList<Head>();

   @Id @GeneratedValue
   public Long getId() {...}
   protected void setId() {...}
   @OneToMany(cascade=CascadeType.ALL)
   public List<Head> getHeads() {
      return Collections.unmodifiableList(heads);
   }
   protected void setHeads() {...}
}

// creates and persists the hydra with 3 heads

// new EntityManager and new transaction
Hydra found = em.find(Hydra.class, hydra.getId());
```

How Many Queries

```
@Entity
public class Hydra {
   private Long id;
   private List<Head> heads = new ArrayList<Head>();

   @Id @GeneratedValue
   public Long getId() {...}
   protected void setId() {...}
   @OneToMany(cascade=CascadeType.ALL)
   public List<Head> getHeads() {
      return Collections.unmodifiableList(heads);
   }
   protected void setHeads() {...}
}

// creates and persists the hydra with 3 heads

// new EntityManager and new transaction
Hydra found = em.find(Hydra.class, hydra.getId());
```

(a) 1 select
(b) 2 selects
(c) 1+3 selects
(d) 2 selects, 1 delete, 3 inserts
(e) None of the above

How Many Queries in 2nd Tx?

(a) 1 select
(b) 2 selects
(c) 1+3 selects
(d) 2 selects, 1 delete, 3 inserts
(e) None of the above

During commit hibernate checks whether the collection property is dirty (needs to be re-created) by comparing Java identities (object references).

IllegalArgumentException is wrapped by PersistentException.

Another Look

```
@Entity
public class Hydra {
   private Long id;
   private List<Head> heads = new ArrayList<Head>();
   @Id @GeneratedValue
   public Long getId() {...}
   protected void setId() {...}
   @OneToMany(cascade=CascadeType.ALL)
   public List<Head> getHeads() {
      return Collections.unmodifiableList(heads);
   }
   protected void setHeads() {...}
}
// new EntityManager and new transaction
Hydra hydra = new Hydra(3);
em.persist(hydra);
// new EntityManager and new transaction
// during find only 1 select (hydra)
Hydra found = em.find(Hydra.class, hydra.getId());
// during commit 1 select (heads),1 delete (heads),3 inserts (heads)
```

Lessons Learned

- Expect unexpected ;-)
- Prefer field access mappings
- Operate on collection objects returned by hibernate
  - Don't change collection references unless you know what you're doing

#4: Where's My Head?

Empty Head

```
@Entity public class Person {
   @Id @GeneratedValue private Long id;
   @Basic private String name;
   @Embedded private Head head = new Head();
   @Embedded private Hand left = new Hand(), right = new Hand():
}
@Embeddable public class Head {
   private String thought;
   public String toString() { return “head”; }
}
@Embeddable public class Hand {
   public String toString() { return “hand”; }
}

Person patrycja = new Person(“patrycja”);
// new EntityManager and new transaction
em.persist(patrycja);
// new EntityManager and new transaction
Person fp = em.find(Person.class, patrycja.getId());
System.out.println(fp.getHead()+“ “+fp.getLeft()+” “+fp.getRight());
found.getLeftHnd() + “ “ + found.getRight);
```

What Does It Print?

(a) head hand hand
(b) head null null
(c) null null null
(d) None of the above

```
@Entity public class Person {
   @Id @GeneratedValue private Long id;
   @Basic private String name;
   @Embedded private Head head = new Head();
   @Embedded private Hand left = new Hand(), right = new Hand():
}
@Embeddable public class Head {
   private String thought;
   public String toString() { return “head”; }
}
@Embeddable public class Hand {
   public String toString() { return “hand”; }
}

Person patrycja = new Person(“patrycja”);
// new EntityManager and new transaction
em.persist(patrycja);
// new EntityManager and new transaction
Person fp = em.find(Person.class, patrycja.getId());
System.out.println(“”+fp.getHead()+“ “+fp.getLeft()+” “+fp.getRight());
found.getLeftHnd() + “ “ + found.getRight);
```

What Does It Print?

(a) head hand hand
(b) head null null
(c) null null null
(d) None of the above

Hibernate does not distinguish between null embedded object and not-null embedded object with all fields null.

Database Dependency

- Some databases (e.g., Oracle) treat **empty strings** as nulls!
- Side effect: your not-null embedded objects with empty strings are retrieved as nulls!

Lessons Learned

- What you store is NOT always what you get!
- Test edge cases
- Test target environments

#5: Lost in Logs

Lost in Logs

```
@Entity public class Application {
   @Id @GeneratedValue private Long id;
   @OneToMany(mappedBy = “app”) Set<Log> logs = new HashSet<Log>();
}
@Entity public class Log {
   @Id @GeneratedValue private Long id;
   Date date;
   String desc;
   @ManyToOne Application app;
   public Log(String desc, Application app) {
      this.date = new Date();
      this.desc = desc;
      this.app = app;
      if (app != null) app.getLogs().add(this);
   }
}
// 10.000 logs for the app inserted
// new EntityManager and new transaction
Application app = em.find(Application.class, appId);
em.persist(new Log(“desc”, app));

//
```

How Many Queries in 2nd

```
@Entity public class Application {
   @Id @GeneratedValue private Long id;
   @OneToMany(mappedBy = “app”) Set<Log> logs = n
}
@Entity public class Log {
   @Id @GeneratedValue private Long id;
   Date date;
   String desc;
   @ManyToOne Application app;
   public Log(String desc, Application app) {
      this.date = new Date();
      this.desc = desc;
      this.app = app;
      if (app != null) app.getLogs().add(this);
   }
}
// one transaction inserts 10.000 logs for the app
// new EntityManager and new transaction
Application app = em.find(Application.class, appId);
em.persist(new Log(“desc”, app));
//
```

(a) 1 select, 1 insert
(b) 2 selects, 1 insert
(c) 1+10000 selects, 1+10001 inserts
(d) None of the above

What Is the Problem?

(a) 1 select, 1 insert
(b) 2 selects, 1 insert
(c) 1+10000 selects, 1+10001 inserts
(d) None of the above

Hibernate must load all 10000 logs into memory to insert a single element to the set (Java set semantics).

How To Fix It?

```
@Entity public class Application {
   @Id @GeneratedValue private Long id;
   @OneToMany(mappedBy = “app”) List<Log> logs = new ArrayList<Log>();
}
@Entity public class Log {
   @Id @GeneratedValue private Long id;
   Date date;
   String desc;
   @ManyToOne Application app;
   public Log(String desc, Application app) {
      this.date = new Date();
      this.desc = desc;
      this.app = app;
      if (app != null) app.getLogs().add(this);
   }
}
// one transaction inserts 10.000 logs for the app
// new EntityManager and new transaction
Application app = em.find(Application.class, appId);
em.persist(new Log(“desc”, app));

//
```

Lessons Learned

- Be careful with large collections in hibernate
- Use List or Collection for the inverse side of a mapping

Conclusion

- Hibernate is reasonably simple and elegant
  - But it has several sharp corners
- Pay more attention to:
  - Transaction management
  - Actual queries executed
- Consider Yonita to detect such issues :-)

Contact

- email: patrycja@yonita.com
- twitter: @yonlabs
- http://www.yonita.com

JUDCon

JBoss Users & Developers Conference

London:2011