What's new in Hibernate

an opinionated cherry pick

Emmanuel Bernard
JBoss by Red Hat

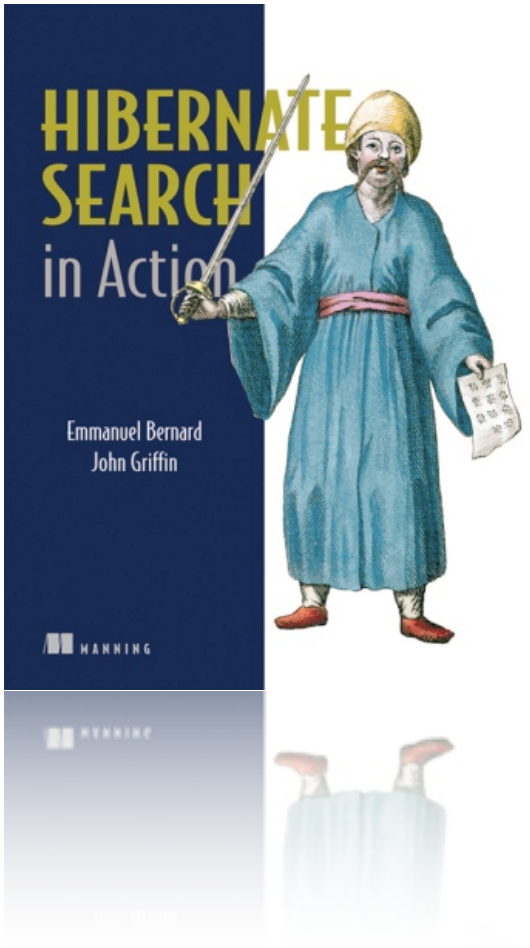
Get an overview of

- what's new in Core
- explore satellite projects

Make you discover new features

Emmanuel Bernard

- Hibernate Search in Action
- blog.emmanuelbernard.com
- twitter.com/emmanuelbernard
- lescastcodeurs.com



Les CastCodeurs

"En Français dans le Code"

http://lescastcodeurs.com

Mapping

All of JPA 2 mapping

- standardization of specific annotations

Some interesting features

Generator - @MapsId

```
@Entity
class Customer {
  @Id UserId userId;

  @MapsId("userId")
  @JoinColumns({
    @JoinColumn(name="userfirstname",
           referencedColumnName="firstName"),
    @JoinColumn(name="userlastname",
           referencedColumnName="lastName")
  })
  @OneToOne User user;
}
```

```
@Entity
class User {
  @EmbeddedId UserId id;
  Integer age;
}

@Embeddable
class UserId implements Serializable
{
  String firstName;
  String lastName;
}
```

Generator - Partial generator

```
@Entity
public class CustomerInventory implements Serializable {
  @Id
  @o.h.a.GenericGenerator(name = "inventory", strategy = "uuid")
  @GeneratedValue(generator = "inventory")
  Integer uuid;

  @Id String location;

  @Id @ManyToOne(cascade = CascadeType.MERGE)
  Customer customer;
}

@Entity
public class Customer implements Serializable {
   @Id
   private int id;
}
```


Generator - @MapsId: a complex case

```
@Entity
class Customer {
  @EmbeddedId CustomerId id;
  boolean preferredCustomer;

  @MapsId("userId")
  @JoinColumns({
    @JoinColumn(name="userfirstname",
                referencedColumnName="firstName"),
    @JoinColumn(name="userlastname",
                referencedColumnName="lastName")
  })
  @OneToOne User user;
}

@Embeddable
class CustomerId implements Serializable {
  UserId userId;
  String customerNumber;
}
```

```
@Entity
class User {
  @EmbeddedId UserId id;
  Integer age;
}

@Embeddable
class UserId implements Serializable
{
  String firstName;
  String lastName;
}
```


Column Read/Write

```
<property name=”creditcard”>
  <column name=”credit_card_num”
          read="decrypt(credit_card_num)"
          write="encrypt(?)"/>
</property>
```


Runtime - fetch profile

Define more than one fetching profile

- classic one `@OneToMany(fetch=EAGER)`
- Some other with specific names

Definition centralized

```
@Entity
@FetchProfile(name = "all",
              fetchOverrides = {
     @FetchProfile.FetchOverride(
         entity = Customer.class,
         association = "orders",
         mode = FetchMode.JOIN)
     @FetchProfile.FetchOverride(
         entity = Order.class,
         association = "country",
         mode = FetchMode.JOIN)
})
public class Customer {
   @Id @GeneratedValue private long id;
   private String name;
   private long customerNumber;
   @OneToMany private Set<Order> orders;

   // standard getter/setter
}

Session session = ...;
session.enableFetchProfile( "all" );  // name matches @FetchProfile name
Customer customer = (Customer) session.get( Customer.class, customerId );
session.disableFetchProfile( "all" ); // or just close the session
```


Criteria API

Object Oriented query building API

Type-safe

Strongly typed

Type-safe

Strongly typed

optionally use a static metamodel

- annotation processor

Static metamodel

```
@Entity
public class Item {
  @Id @GeneratedValue public Long getId() {}
  public Boolean isShipped() {}
  public String getName() {}
  public BigDecimal getPrice() {}
  @OneToMany public Map<String, Photo> getPhotos() {}
  public Order getOrder() {}
  public Product getProduct() {}
}

@StaticMetamodel(Item.class)
public class Item_ {
  public static SingularAttribute<Item, Long> id;
  public static SingularAttribute<Item, Boolean> shipped;
  public static SingularAttribute<Item, String> name;
  public static SingularAttribute<Item, BigDecimal> price;
  public static MapAttribute<Item, String, Photo> photos;
  public static SingularAttribute<Item, Order> order;
  public static SingularAttribute<Item, Product> product;
}
```


How to build a query

From EntityManager

CriteriaBuilder is a helper class

CriteriaQuery represents your query

Type safe

```
EntityManager em;
CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<Order> critQ = qb.createQuery(Order.class);

[...]

TypedQuery<Order> q = em.createQuery(critQ);
List<Order> orders = q.getResultList();
```


Fetching

```
//select c from Customer c
Root<Customer> customer = cq.from(Customer.class);
```

```
//select c from Customer c join fetch c.orders
cq.from(Customer.class)
     .fetch(Customer_.orders, INNER);
```


Joins and Where

CriteriaBuilder is a function placeholder

```
SELECT c.name
FROM Customer c JOIN c.orders o JOIN o.items i
WHERE i.product.price > 200
```

```
Root<Customer> c = cq.from(Customer.class);
Path<Order, Item> i = c.join(Customer_.orders).join(Order_.items);

cq.select( c.get(Customer_.name) )
  .where(
    cb.greaterThan(
        i.get(Item_.product).get(Product_.price) ),
          200 )
        );
```

Having and count

```
SELECT c.status, AVG(c.filledOrderCount), COUNT(c)
FROM Customer c
GROUP BY c.status
HAVING c.status IN (1, 2)
```

```
Root<Customer> cust = cq.from(Customer.class);

q.select( cust.get(Customer_.status),
          cb.avg(cust.get(Customer_.filledOrderCount),
          cb.count(cust) )
  .group( cust.get(Customer_.status) )
  .having( cb.in( cust.get(Customer_.status) )
                 .value(1)
                 .value(2) );
```


Subquery

```
SELECT DISTINCT emp
 FROM Employee emp
 WHERE EXISTS (
   SELECT spouseEmp
   FROM Employee spouseEmp
   WHERE spouse = emp.spouse)

Root<Employee> emp = cq.from(Employee.class);

Subquery<Employee> subQuery = cq.subquery(Employee.class);

Root<Employee> spouse = subQuery.from(Employee.class);
subQuery.where(cb.equal(spouse,emp.get(Employee_.spouse));

cq.distinct(true).where( cb.exists( subQuery ) );
```

Support *all* of JP-QL

Collections and maps

Aggregation, order by, group by

Having, count

Subselect

Lock Mode

- Prevents
  - dirty reads
  - non-repeatable reads
- Types
  - Optimistic
  - Pessimistic Read / Write
- Force increment
- Make use of Database locks in a standard way

Packaging

Core is made of several modules including

- annotations
- entitymanager
- envers
- ... core :)

Easier for your to chose what you want

- doc on the way

Other projects

Hibernate Search

Full-text search for Hibernate application

Solve mismatches

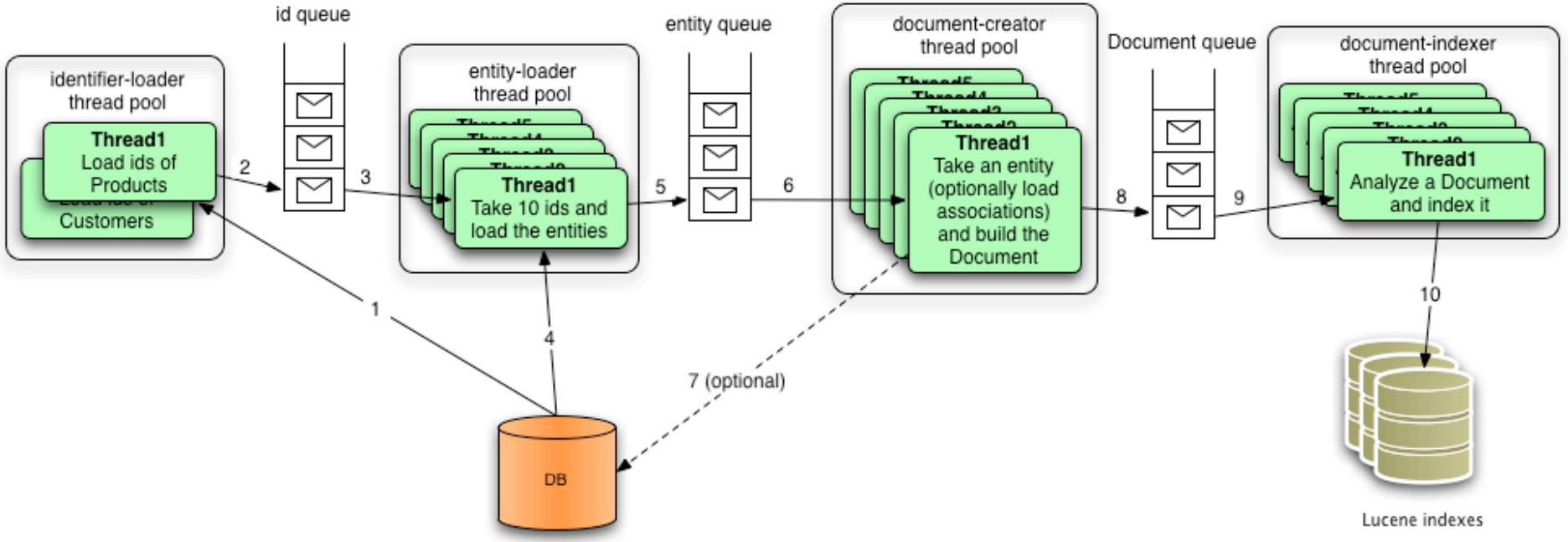
- Transparent index synchronization
- Object model conversion
- Unified programmatic model
  - Criteria, HQL, SQL, full-text

Clustering

Massive indexing

```
fullTextSession.createIndexer().startAndWait();

fullTextSession
   .createIndexer( User.class )
   .batchSizeToLoadObjects( 25 )
   .cacheMode( CacheMode.NORMAL )
   .threadsToLoadObjects( 5 )
   .threadsForSubsequentFetching( 20 )
   .startAndWait();
```



id queue
entity queue
document-creator thread pool
Document queue
document-indexer thread pool
identifier-loader thread pool
Thread1
Load ids of Products
Customers
entity-loader thread pool
Thread1
Take 10 ids and load the entities
Thread1
Take an entity (optionally load associations) and build the Document
Thread1
Analyze a Document and index it
1
2
3
4
5
6
7 (optional)
8
9
10
DB
Lucene indexes

Programmatic API

```
SearchMapping mapping = new SearchMapping();
      mapping.analyzerDef( "stem", StandardTokenizerFactory.class )
                .tokenizerParam( "name", "value" )
                .tokenizerParam(  "name2", "value2" )
                .filter( LowerCaseFilterFactory.class )
                .filter( SnowballPorterFilterFactory.class)
                   .param("language", "English")
             .entity(Address.class).indexed().indexName("Address_Index")
                .property("street1", ElementType.FIELD)
                   .field()
                   .field()
                      .name("street1_iso")
                      .store( Store.YES )
                      .index( Index.TOKENIZED )
                      .analyzer( ISOLatin1Analyzer.class)
                   .field()
                      .name("street1_ngram")
                      .analyzer("ngram")
             .entity(User.class).indexed()
                .property("name", ElementType.METHOD)
                   .field();
```


More

Error handling

- report to queue or log

JGroups clustering

Infinispan-backed Lucene Directory

Hibernate Envers

Store audit information transparently

Historical data

Revision is global like SVN

The existing table schema is unchanged

Person:

Id	Name	Surname
123	John	Smith
857	Brad	Pitt
698	Mary	Doe
	...	

Person_AUD:

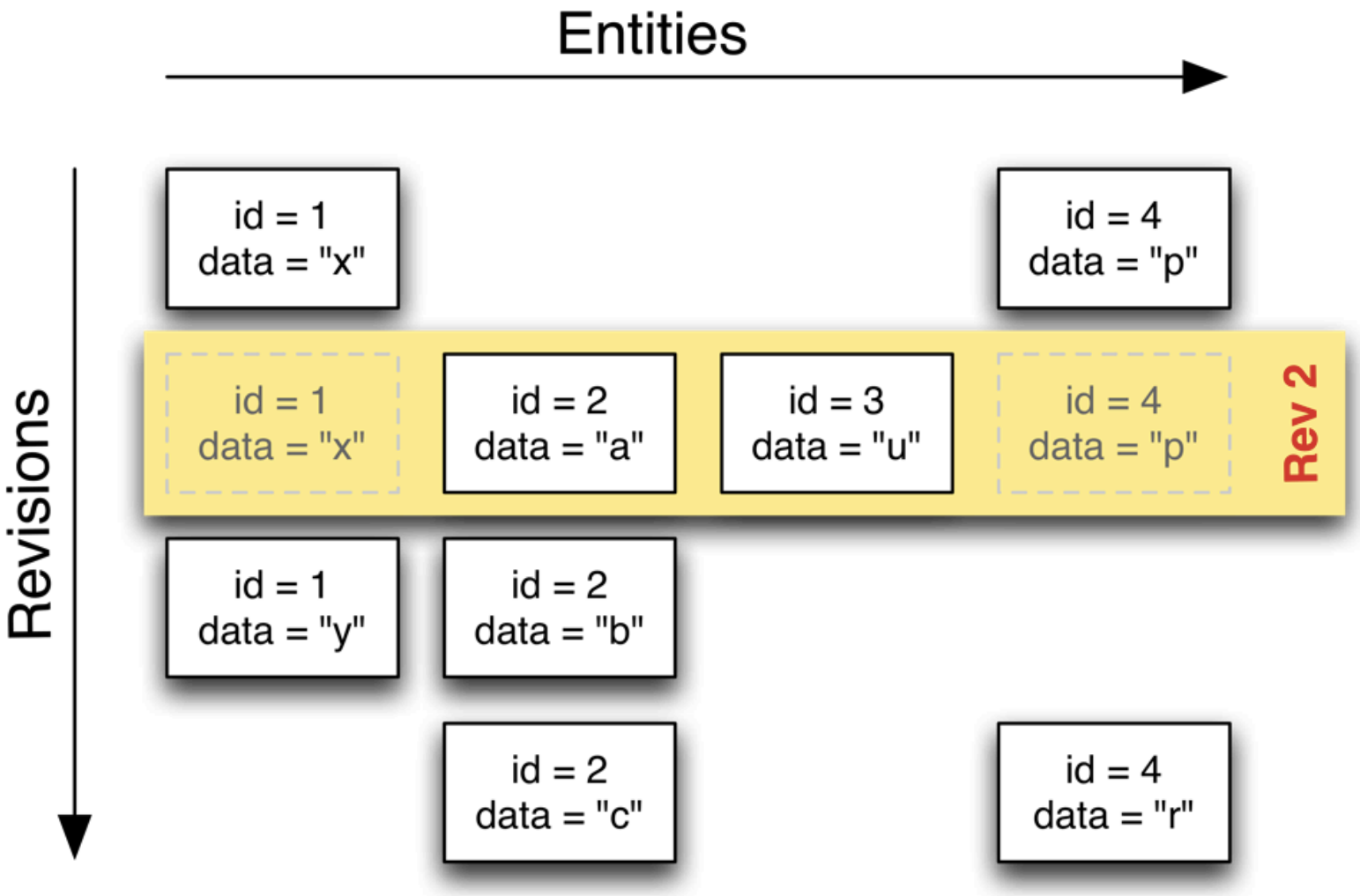
Rev number	Id	Name	Surname	Rev type
64	123	James	Smith	**ADD**
64	857	Brad	Pitt	**ADD**
85	123	Peter	Smith	**MOD**
90	501	Arnold	Schwarzenegger	**DEL**
90	123	John	Smith	**MOD**

`@Audited`

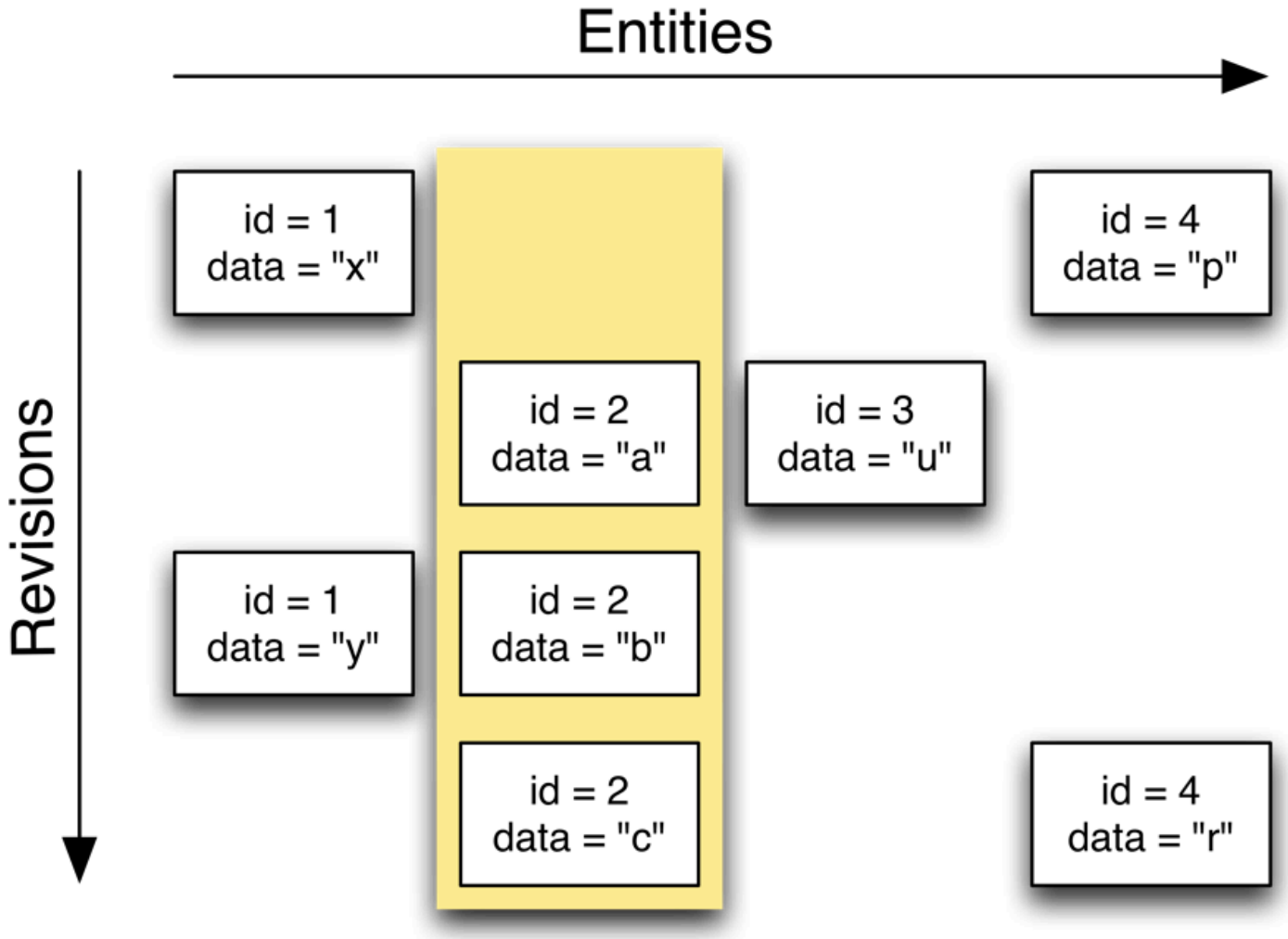
Lookup by revision

including navigation

Query per revision



Query by entity history



Hibernate Validator

- Bean Validation RI
- Describe constraint on domain model

```
@Entity
public class Customer {
   @CustomerNumber @NotNull
   public long getCustomerNumber() { return customerNumber };
}
```

- Automatically validated
  - on presentation layer (JSF 2, Wicket and more)
  - on JPA entity persist, update, remove
  - table schema reflecting constraints
- Manually via API

To sum up

Core new features

- JPA 2
- fetch profiles
- partial generators
- ...

Hibernate Search: full-text search your entities

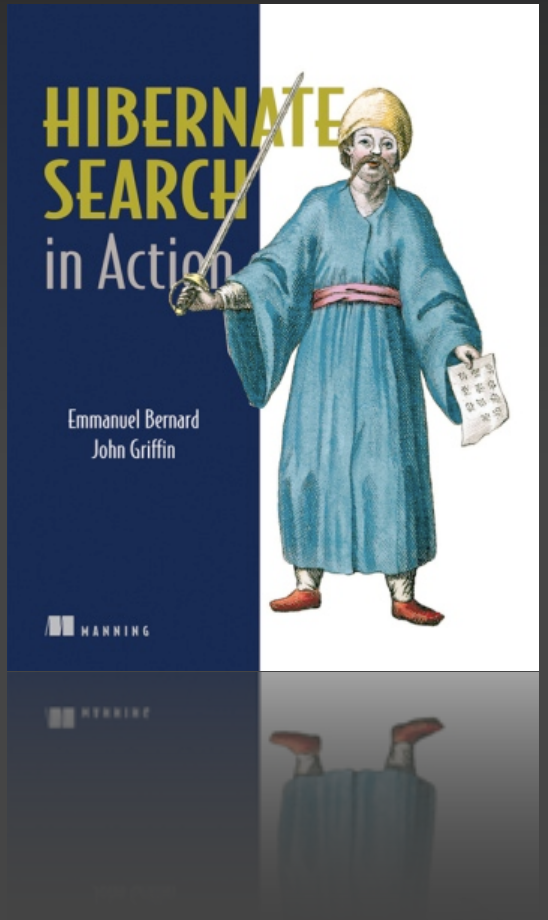
Envers: auditing your entities

Hibernate Validator: declarative constraint and validation

Questions?

http://hibernate.org

http://in.relation.to



FOLLOW US ON TWITTER

www.twitter.com/redhatsummit

TWEET ABOUT IT

#summitjbw

READ THE BLOG

http://summitblog.redhat.com/