Envers
Easy Entity Versioning Auditing

Adam Warski
Software Engineer
Level N Consulting

www.devoxx.com
Who am I?

- Adam Warski :)
- Formerly at JBoss
- Senior Software Engineer at Level N Consulting
- Creator of Envers
- [http://www.warski.org/blog](http://www.warski.org/blog) (Envers, Seam, Typestate, ...)


Agenda

- Overview of patterns for auditing
- What is Envers?
- How does it work?
- Configuration
- An example
- Queries
- Use cases
- Using Envers
Overview of patterns for auditing

What’s the problem and how to approach it?
Example: Person & Address

- String name
- String surname

- String streetName
- Integer houseNumber

We want to store the history of a Person's addresses
Patterns: Audit Log

- Logging all activity to a file or database
- Entity state: String or CSV
- Very simple
- Hard to read historic data

2008-08-10 06:24:19,761 [org.jboss.envers.example.Person] add,id=1,name=John,surname=Doe,address=10
2008-08-25 18:49:18,420 [org.jboss.envers.example.Person] mod,id=1,name=John,surname=Doe,address=24
2008-09-10 13:44:42,120 [org.jboss.envers.example.Address] add,id=29,streetName=East st.,houseNumber=53
2008-09-16 01:12:33,590 [org.jboss.envers.example.Person] mod,name=John,surname=Doe,address=29

~
Patterns: Effectivity

- Explicit start and end date (validity)
- Verbose
- “Manual”
- Complicates mapping

Person
- String name
- String surname

Address
- String streetName
- Integer houseNumber
- Date validFrom
- Date validTo
Patterns: **Temporal property**

- Getter with a Date argument
- Complicates mapping; facades needed
- Verbose

When more properties are temporal: temporal object / snapshot

---

**Person**
- String name
- String surname
- Address getAddress(Date validOn)

**Address**
- String streetName
- Integer houseNumber
Patterns overall

In most use cases and most of the time:

- we are only interested in “current” data

Historical data:

- much less often
- in different places - uniform access not a necessity
Envers

What is it and how does it work?
What is Envers?

- An entity auditing (versioning) library
- Part of Hibernate
- Simplifies storing and retrieving historical data
Assumptions

- **Transparent**: data can be used as always (queried, persisted, etc.)

- **Not intrusive**: 
  - The database schema isn’t changed (some tables can be added)
  - Minimal code changes

- **Slowly changing data**
How does Envers work?

- The programmer specifies which entities should be audited.
- For each audited entity, an audit entity is (dynamically) created.
- E.g. entity “Address” has an “Address_AUD” companion.
- The companion stores historical data.
How does Envers work?

- On an update/insert/delete: data inserted to audit tables
- All changes in a transaction: 1 revision
- Revisions capture consistent state
- Revisions are global
- Similar to SVN
Envers Configuration

- To make an entity audited: annotate with `@Audited`
- Add event listeners to `persistence.xml`

```xml
<property name="hibernate.ejb.event.post-insert" value="org.hibernate.envers.event.AuditEventListener" />
<property name="hibernate.ejb.event.post-update" value="org.hibernate.envers.event.AuditEventListener" />
<property name="hibernate.ejb.event.post-delete" value="org.hibernate.envers.event.AuditEventListener" />
<property name="hibernate.ejb.event.pre-collection-update" value="org.hibernate.envers.event.AuditEventListener" />
<property name="hibernate.ejb.event.pre-collection-remove" value="org.hibernate.envers.event.AuditEventListener" />
<property name="hibernate.ejb.event.post-collection-recreate" value="org.hibernate.envers.event.AuditEventListener" />
```

```java
@Entity
public class Person { 
    @Id
    @GeneratedValue
    private int id;

    @Audited
    private String name;

    @Audited
    @ManyToMany
    private Address address;
```
What can be audited?

- Mappings defined by JPA:
  - Simple properties: Strings, Integers, Dates, ...
  - Components
  - Relations

- Some Hibernate extensions
  - Custom types
  - Collections
Person & Address example

- String name
- String surname

- String streetName
- Integer houseNumber

• Now let's assume that both entities are annotated with @Audited
em.getTransaction().begin();
Address a1 = new Address("West st.", 10);
Address a2 = new Address("East st.", 15);
Person p = new Person("John", "Doe");
p.setAddress(a1);
entityManager.persist(a1);
entityManager.persist(a2);
entityManager.persist(p);
em.getTransaction().commit();
// Revision 2

em.getTransaction().begin();

p = entityManager.find(Person.class, id);

p.setName("Paul");

p.setAddress(a2);

em.getTransaction().commit();

- Old person data is stored

- No code changes - completely transparent to the user
AuditReader ar = AuditReaderFactory.get(em);

// Reading the person at revision 1
old_p = ar.find(Person.class, id, 1);
assert "John".equals(old_p.getName());
assert a1.equals(old_p.getAddress());
// Reading the addresses at revision 1

old_a1 = ar.find(Address.class, a1_id, 1);
assert old_a1.getPersons().size() == 1;
assert old_a1.getPersons().contains(p);

old_a2 = ar.find(Address.class, a2_id, 1);
assert old_a2.getPersons().size() == 0;

- Transparent traversing of relations: also in case of collections
Querying

How to query historical data?
Querying

- Entities-at-revision
- Revisions-of-entity
- Inspired by criteria queries

```java
auditReader.createQuery().
  .forRevisionsOfEntity(Person.class, false, true)
  .addProjection(AuditEntity.revisionNumber().count())
  .add(AuditEntity.id().eq(person.getId()))
  .getSingleResult()
```
Logging data for revisions

- With each revision, arbitrary data can be bound (metadata)
- For example: user making the changes

- Special entity (@RevisionEntity), storing the metadata
- Listener, invoked when a new revision is created
Use-cases

Envers in practice
Use case: Structured Wiki

- The power of wikis: everybody can edit
- It works well, because history is stored and viewable by everyone
- Plus, you know who made the changes

What if the editable part of your website is more than just one textbox?

- e.g. a set of links, images
Step 1: main entity

```java
@Entity
@Audited

public class WikiPage {

    @Id @GeneratedValue private Long id;

    private String title;

    private String content;

    @CollectionOfElements private Set<String> links;

    @OneToMany private Set<Image> images;
}
```
Step 2: revision entity

@Entity

@RevisionEntity(WikiListener.class)

public class WikiRevision {

    @Id @GeneratedValue @RevisionNumber
    private Long id;

    @RevisionTimestamp private Long timestamp;

    @ManyToOne private User modifiedBy;

    // (...)
public class WikiListener implements RevisionListener {

    public void newRevision(Object revEntity) {

        WikiRevision wikiRev = (WikiRevision) revEntity;

        User currentUser = (User) Component.getInstance("currentUser");

        wikiRev.setModifiedBy(currentUser);

    }

}
public List getHistory(int from, int count, Long pageId) {

    return auditReader.createQuery()
    .forRevisionsOfEntity(WikiPage.class, false, true)
    .addOrder(revisionNumber().desc())
    .add(id().eq(pageId))
    .setFirstResult(from)
    .setMaxResults(count)
    .getResultList();
}
public List getChangesByUser(User user) {
    return auditReader.createQuery()
        .forRevisionsOfEntity(WikiPage.class, false, true)
        .addOrder(revisionNumber().desc())
        .add(revisionProperty("modifiedBy").eq(user))
        .getResultList();
}
We add a field to `WikiPage`: `verified`

We want to find the latest verified version of a page

```java
auditReader.createQuery()
    .forRevisionsOfEntity(WikiPage.class, false, true)
    .add(revisionNumber().maximize())
    .add(property("verified").eq(true))
    .add(id().eq(pageId))
    .getResultList();
```
Practical
How to use Envers?

- Works everywhere where Hibernate works
  - Standalone, Webapps, Seam, Spring, ...

- Formerly a stand-alone project
  - 1.1.0.GA released in October

- Now part of Hibernate
  - Next release in 1-2 months
How to use Envers?

Download from:

http://www.jboss.org/envers

Also in JBoss Snapshot repository:

http://www.jboss.org/community/docs/DOC-11381
Your data is safe!

**Person:**

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Surname</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>857</td>
<td>Brad</td>
<td>Pitt</td>
</tr>
<tr>
<td>698</td>
<td>Mary</td>
<td>Doe</td>
</tr>
</tbody>
</table>

**Person_AUD:**

<table>
<thead>
<tr>
<th>Rev number</th>
<th>Id</th>
<th>Name</th>
<th>Surname</th>
<th>Rev type</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>123</td>
<td>James</td>
<td>Smith</td>
<td>ADD</td>
</tr>
<tr>
<td>64</td>
<td>857</td>
<td>Brad</td>
<td>Pitt</td>
<td>ADD</td>
</tr>
<tr>
<td>85</td>
<td>123</td>
<td>Peter</td>
<td>Smith</td>
<td>MOD</td>
</tr>
<tr>
<td>90</td>
<td>501</td>
<td>Arnold</td>
<td>Schwarzenegger</td>
<td>DEL</td>
</tr>
<tr>
<td>90</td>
<td>123</td>
<td>John</td>
<td>Smith</td>
<td>MOD</td>
</tr>
</tbody>
</table>
### Performance

- Must be slower than without auditing:
  - 1 insert for each modified entity
  - 1 insert for each transaction

<table>
<thead>
<tr>
<th>MySQL 5.1.30 (InnoDB)</th>
<th>5000 inserts</th>
<th>1000 complex inserts</th>
<th>5000 updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not audited</td>
<td>6.307s</td>
<td>6.622s</td>
<td>8.487s</td>
</tr>
<tr>
<td>Audited</td>
<td>9.807s</td>
<td>12.758s</td>
<td>11.444s</td>
</tr>
<tr>
<td>Difference</td>
<td>x 1.55</td>
<td>x 1.92</td>
<td>x 1.34</td>
</tr>
</tbody>
</table>
Envers overall

- Unchanged mapping (not intrusive)
- Unchanged code (transparent)
- Straightforward history reading
- Deleted entities aren't gone
- Easier to use (@Audited)
- Data for revisions
- Queries
Future

- Support for other Hibernate-specific mappings
  - relations in components
  - collections of components

- JPA2
Future

- Tools
  - import
  - revert
  - branch

- DIFF

- Different auditing strategies
  - storing only fields
Future

Tagging

- a tag can be anything
- only one revision can be tagged (many?)
- find entity by tag
Thank you for your attention!

http://www.jboss.org/envers/
adam@warski.org