



Envers

Proste Wersjonowanie Encji

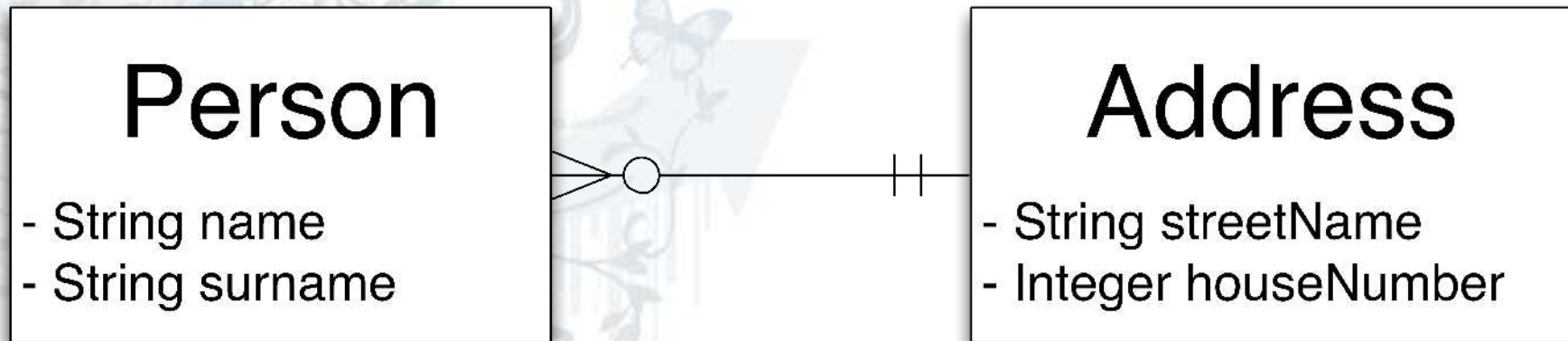
Adam Warski
JBoss / Red Hat
adam@warski.org

Java Developers' Day, 16 października 2008, Kraków

Agenda

- Przegląd wzorców do wersjonowania
- Co to jest Envers?
- Jak działa?
- Konfiguracja
- Przykład
- Zapytania
- Przypadek użycia
- Status projektu

Przykład: osoba i adres



- Chcemy pamiętać historię adresów danej osoby

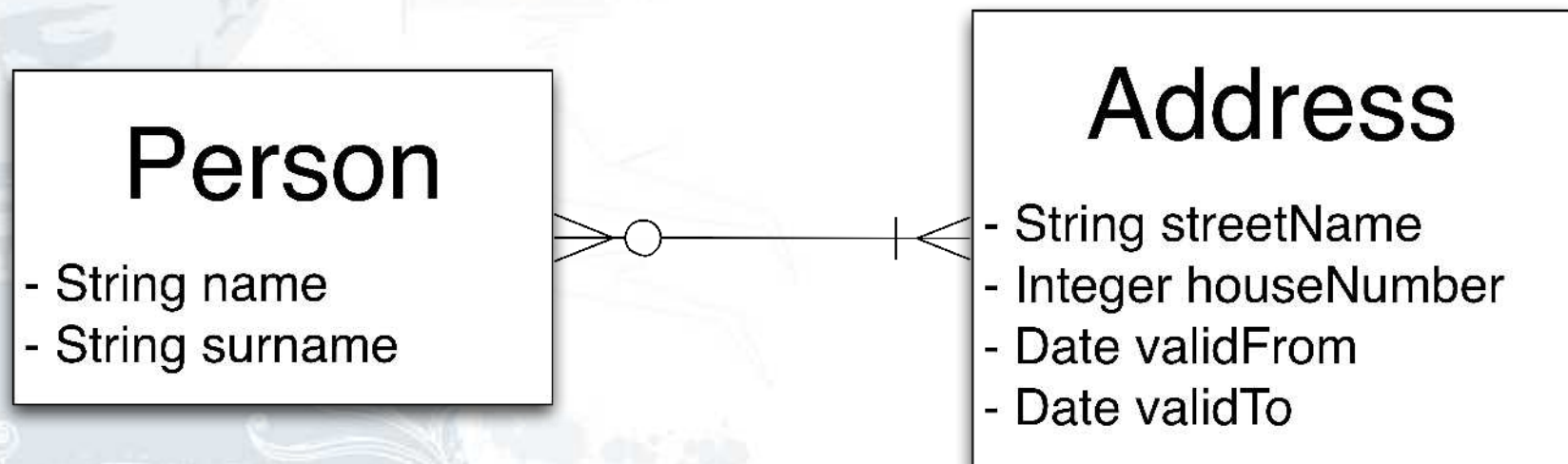
Wzorce: Audit Log

- Wszystkie zdarzenia logowane są w pliku lub bazie danych
- Stan encji: String lub CSV
- Bardzo prosty koncepcyjnie
- Trudno odczytać historyczne dane

```
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
```

Wzorce: Effectivity

- Daty początku i końca ważności encji
- Komplikuje użycie
- Komplikuje mapowanie
- Usunięte encje znikają z bazy



Wzorce: Temporal property

- Getter* z argumentem typu Date
- Komplikuje mapowanie; potrzebne fasady
- Komplikuje użycie
- Gdy więcej własności temporalnych:
temporal object / snapshot

Person

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

Address

- String streetName
- Integer houseNumber

Wzorce ogólnie

- W większości przypadków jesteśmy zainteresowani tylko “aktualnymi” danymi
- Dane historyczne:
 - używane dużo rzadziej
 - w innych miejscach – jednolity dostęp nie jest konieczny

Co to jest Envers?

- Biblioteka do wersjonowania encji (**entity versioning**)
- Współpracuje z Hibernate i JPA
- Ułatwia zapamiętywanie i odczytywanie historycznych danych

Założenia

- **Przezroczystość**: dostęp do “aktualnych” danych bez zmian (zapamiętywanie, usuwanie, zapytania, itd.)
- **Nieinwazyjność**:
 - Schemat bazy danych się nie zmienia (dodajemy tylko nowe tabele)
 - Minimalne zmiany w kodzie
- **Powoli** zmieniające się dane

Jak działa Envers?

- Programista wskazuje, które encje będą wersjonowane
- Dla każdej wersjonowanej encji, dodawana jest encja wersji
- np. dla encji “Address” utworzona zostanie encja “Address_versions”
- Dodatkowa encja przechowuje dane historyczne

Jak działa Envers?

- Przy zmianie/dodaniu/usunięciu encji: dane dodawane do encji wersji
- Wszystkie zmiany w transakcji: 1 rewizja
- Rewizje ujmują spójny stan
- Rewizje są globalne
- Podobnie do SVN-a

Envers – Konfiguracja

- Aby encja była wersjonowana: dodajemy anotację **@Versioned**
- Dodajemy event listeners do `persistence.xml`

```
<!-- Envers -->
<property name="hibernate.ejb.event.post-insert"
  value="org.jboss.envers.event.VersionsEventListener" />
<property name="hibernate.ejb.event.post-update"
  value="org.jboss.envers.event.VersionsEventListener" />
<property name="hibernate.ejb.event.post-delete"
  value="org.jboss.envers.event.VersionsEventListener" />
```

```
/**
 * @author Adam Warski (adam at
 */
@Entity
@Versioned
public class Person {
    @Id
    @GeneratedValue
    private int id;

    private String name;

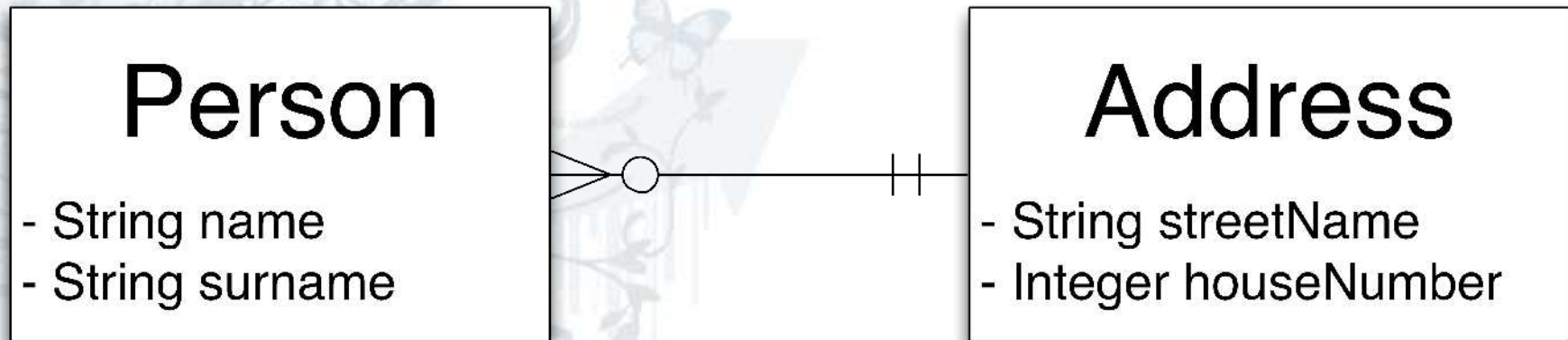
    private String surname;

    @ManyToOne
    private Address address;
```

Co może być wersjonowane?

- Proste właściwości : napisy (`String`), liczby (`Integer`), daty (`Date`), ...
- Komponenty
- Zbiory, listy, mapy
- Relacje

Osoba i adres: przykład ponownie



- Załóżmy, że obie encje mają anotację **@Versioned**

```
// Rewizja 1
em.getTransaction().begin();
Address a1 = new Address("Puławska", 10);
Address a2 = new Address("Wilanowska", 15);
Person p = new Person("Jan", "Kowalski");
p.setAddress(a1);
entityManager.persist(a1);
entityManager.persist(a2);
entityManager.persist(p);
em.getTransaction().commit();
```

```
// Rewizja 2
em.getTransaction().begin();
p = entityManager.find(Person.class, id);
p.setName("Zbigniew");
p.setAddress(a2);
em.getTransaction().commit();
```

- Stare dane osoby są zapamiętane
- Brak zmian w kodzie – przezroczystość

```
VersionReader vr =  
    VersionReaderFactory.get(em);  
// Czytamy osobę w rewizji 1  
old_p = vr.find(Person.class, id, 1);  
assert "Jan".equals(old_p.getName());  
assert a1.equals(old_p.getAddress());
```

- Przezroczyste czytanie relacji

```
// Czytamy adres w rewizji 1
old_a1 = vr.find(Address.class, a1_id, 1);
assert old_a1.getPersons().size() == 1;
assert old_a1.getPersons().contains(p);

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

- Przezroczyste czytanie relacji: też w przypadku kolekcji

Twoje dane są bezpieczne!

Page:

Id	Title	Content
123	Title 1	Content 1
857	Title 2	Content 2
698	Title 3	Content 3
...		

Page_versions:

Rev number	Id	Title	Content	Rev type
64	123	Oldest title 1	Oldest content 1	ADD
64	857	Title 2	Content 2	ADD
85	123	Older title 1	Older content 1	MOD
90	501	Title 4	Content 4	DEL
90	123	Title 1	Content 1	MOD

Zapytania

Encje

Rewizje

id = 1
data = "x"

id = 4
data = "p"

id = 2
data = "a"

id = 3
data = "u"

id = 1
data = "y"

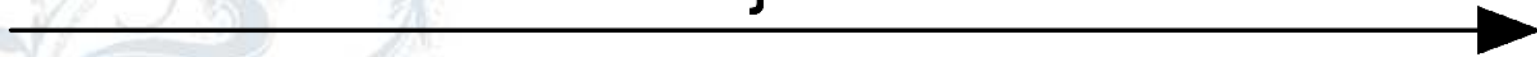
id = 2
data = "b"

id = 2
data = "c"

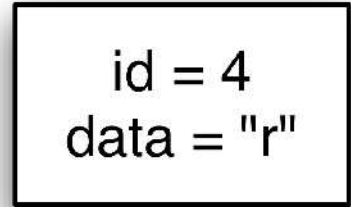
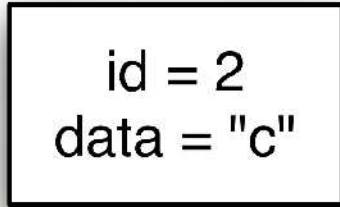
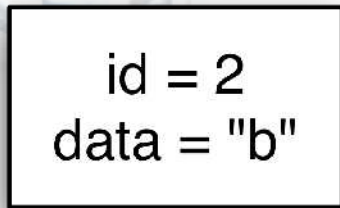
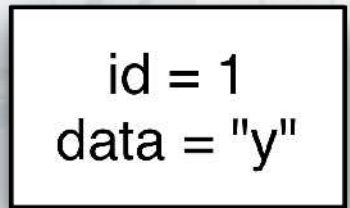
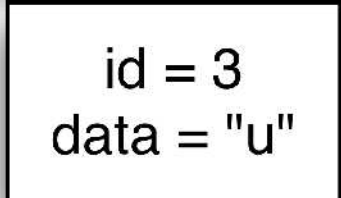
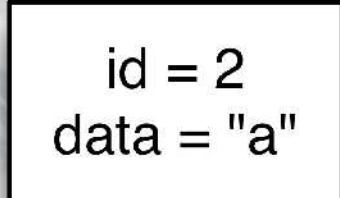
id = 4
data = "r"

Zapytania

Encje

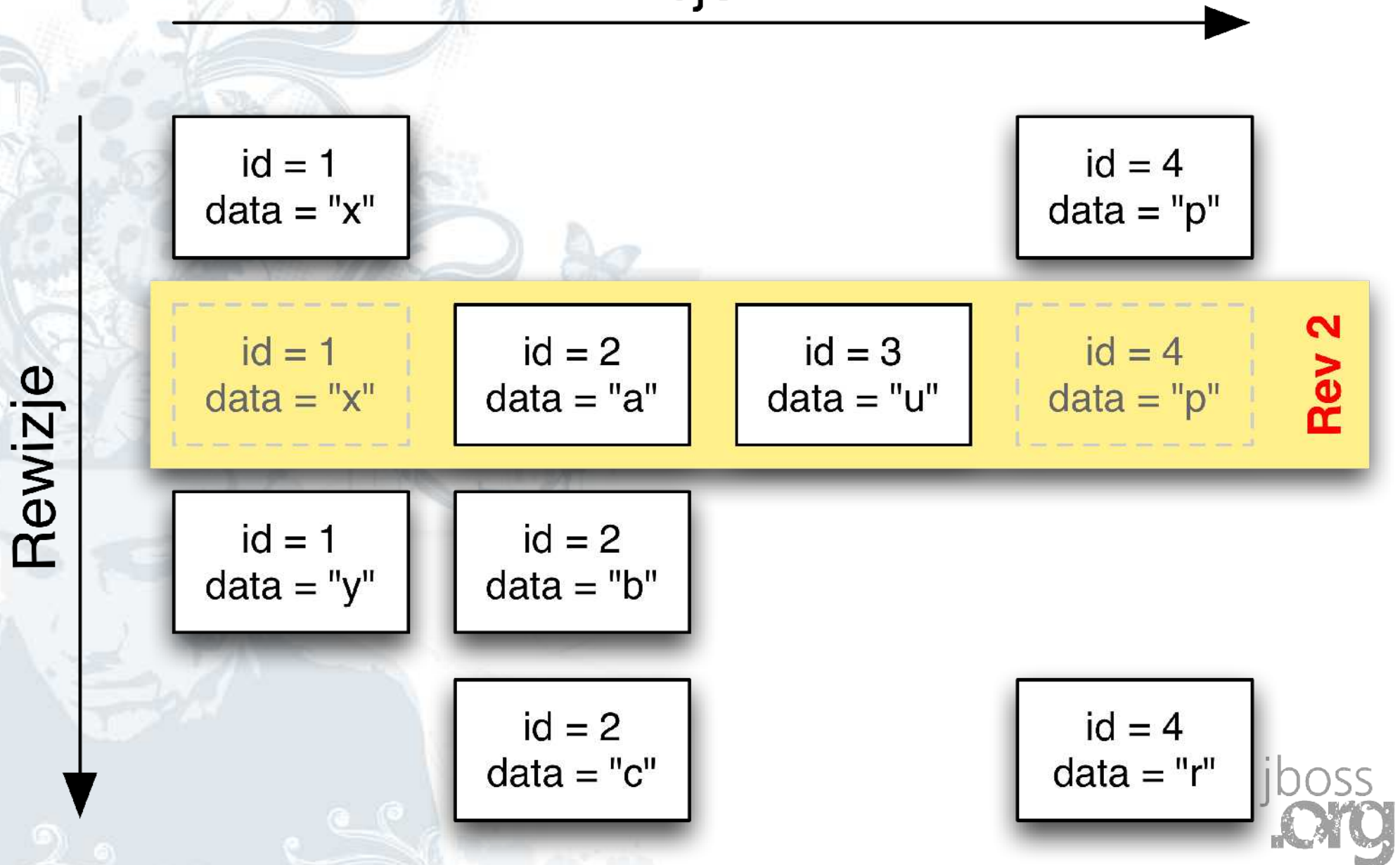


Rewizje



Zapytania

Encje



Zapytania

Encje

Rewizje

id = 1
data = "x"

id = 1
data = "y"

id = 2
data = "a"

id = 2
data = "b"

id = 2
data = "c"

id = 3
data = "u"

id = 4
data = "p"

id = 4
data = "r"

Zapytania

- Encje w danej rewizji (“Entities-at-revision”)
- Rewizje encji (“Revisions-of-entity”)
- API zainspirowane criteria queries

```
versionsReader.createQuery()  
  .forRevisionsOfEntity(Person.class, false, true)  
  .addProjection(RevisionProperty.count())  
  .add(VersionsRestrictions.idEq(person.getId()))  
  .getSingleResult()
```

Dodatkowe dane dla rewizji

- Z każdą rewizją możemy powiązać dowolne dane (**metadane**)
- Na przykład: użytkownik dokonujący zmian
- **Specjalna encja** (**@RevisionEntity**), przechowująca metadane
- **Listener**, wywoływany gdy tworzona jest nowa encja rewizji

Przypadek użycia: Strukturalne Wiki

- Główna zaleta Wiki: wszyscy mogą edytować
- Działa dobrze, bo:
 - Przechowywana jest historia
 - Wiadomo kto dokonał zmian
- Co jeżeli edytowalna część Twojej strony to więcej niż jedno pole tekstowe?
 - np. zbiór odnośników, obrazków

Krok 1: główna encja

@Entity

@Versioned

```
public class WikiPage {  
    @Id @GeneratedValue private Long id;  
    private String name;  
    private String description;  
    @CollectionOfElements  
        private Set<String> links;  
    @OneToMany private Set<Image> images;  
    // Getters, setters, equals, hashCode  
}
```

Krok 2: encja rewizji

```
@Entity
@RevisionEntity(WikiListener.class)
public class WikiRevision {
    @Id @GeneratedValue
        @RevisionNumber private Long id;
        @RevisionTimestamp private Long timestamp;

    @ManyToOne private User modifiedBy;

    // Getters, setters, equals, hashCode
}
```

Krok 3: listener nowej rewizji

```
public class WikiListener implements
RevisionListener {
    public void newRevision(Object revEntity) {
        WikiRevision wikiRev = (WikiRevision)
            revEntity;
        User currentUser = (User)
            Component.getInstance("currentUser");
        wikiRev.setModifiedBy(currentUser);
    }
}
```

Krok 4: przeglądanie historii

```
public List getHistory(int from, int count,
    Long pageId) {
    return versionsReader.createQuery()
        .forRevisionsOfEntity(WikiPage.class, false,
            true)
        .addOrder(RevisionProperty.desc())
        .add(idEq(pageId))
        .setFirstResult(from)
        .setMaxResults(count)
        .getResultList();
}
```

Wydajność

- **Musi** być wolniej niż bez wersjonowania:
 - 1 wiersz: każda modyfikowana encja
 - 1 wiersz: transakcja

	1000 x update	1000 x insert
MySQL 5.0.3P (MyISAM)	44s vs) s (2.18)	4s vs 1 4s (2.05)
PostgreSQL 8.3.4	48s vs 102s (2.10)	8s vs 1 s (2.01)

Testy wykonane na MacBooku Pro 2.16GHz, 2GB RAM

Podsumowanie: Envers

- Brak zmian w **mapowaniu** (nieinwazyjność)
- Brak zmian w **kodzie** (przezroczystość)
- Proste **czytanie historii**
- **Usunięte encje** nie znikają
- **Łatwo użyć** (**@Versioned**)
- Dodatkowe dane dla rewizji
- **Zapytania**

Envers: status projektu

- 1.0.0.GA wypuszczone w czerwcu, 1.1.0.beta2 na początku października
- 308 testów, 69% pokrycie kodu
- W przyszłości:
 - wsparcie dla innych strategii wersjonowania
 - pełne wsparcie dla dziedziczenia (joined i table-per-class)
 - narzędzia (import, przywracanie, gałęzie?)



Dziękuję!

Pytania?

<http://www.jboss.org/envers/>

adam@warski.org