Vaak geldt als functionele eis aan applicaties dat achteraf kan worden gezien wie, wat, wanneer heeft gedaan: een audittrail. Soms geldt ook (vanuit wettelijke gronden) de sterkere eis dat het mogelijk moet zijn de gegevens terug te halen zoals die op een specifiek moment in het verleden waren. Dit artikel laat zien hoe aan deze eisen kan worden voldaan met Hibernate Envers.

Auditing met Hibernate Envers

Wijzigingen bijhouden zonder eigen code

Frans van Buul
is SOA/Java software architect bij Inter Access. Hij is bereikbaar via frans.van.buul@interaccess.nl.

Envers is een open source auditing framework dat is gestart door Adam Warski. Na de eerste productierelease in juli 2008 is Envers in oktober van dat jaar een officiële module geworden binnen Hibernate. Het heeft nu een vergelijkbare status als Hibernate Search, de Hibernate module die full-text search mogelijk maakt.
Op hoofdlijnen kun je de werking van Envers vergelijken met een reeds langer bestaande techniek die hier ook voor wordt gebruikt: het bijhouden van wijzigingen op de inhoud van databasetabellen middels triggers. Omdat Envers op JPA-niveau werkt in plaats van op databaseniveau, biedt het meer mogelijkheden: je kunt bijvoorbeeld ook historische JPA queries uitvoeren.

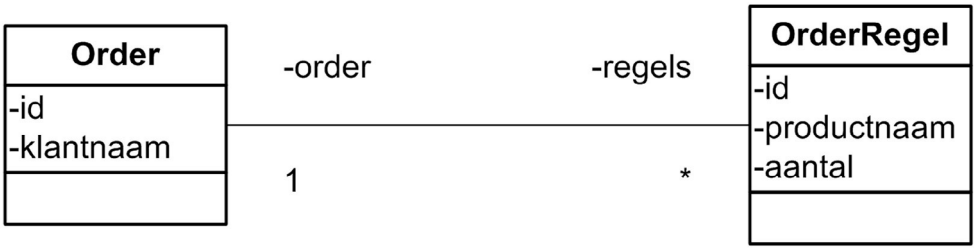
Een voorbeeldcase

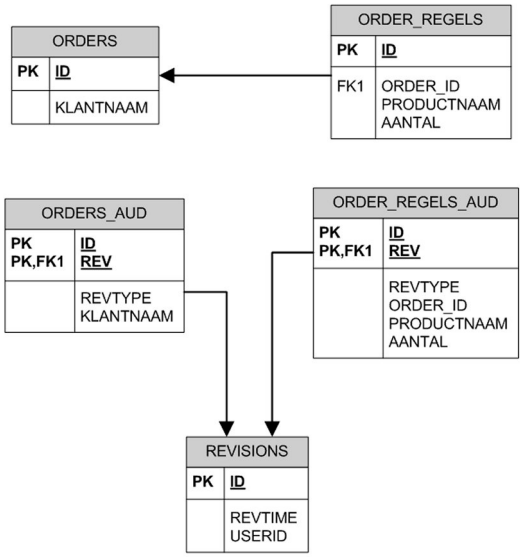
In dit artikel verkennen we de mogelijkheden van Envers aan de hand van een voorbeeld dat eenvoudig is, maar desondanks de essentiële punten raakt. Van dit voorbeeld is ook werkende code beschikbaar die via de auteur kan worden verkregen. Het voorbeeld is een fragment van een denkbeeldige applicatie die bestellingen verwerkt, waarbij één Order meerdere OrderRegels kan bevatten. Een OrderRegel bevat informatie om welk product het gaat en hoeveel eenheden van dat product worden besteld. Hieronder is het class diagram weergegeven.

Het bijhouden van wijzigingen

Envers slaat wijzigingen in 'audited' entities op in de database. Entities zijn audited wanneer de class is voorzien van de annotatie *@Audited*. Envers heeft voor elke audited entity class, naast de 'gewone' JPA-tabel ook een corresponderende audittabel nodig. Deze audittabellen verwijzen naar één overkoepelende tabel waarin gegevens over een wijziging als geheel worden opgeslagen. Een databaseschema voor ons voorbeeld is weergegeven in figuur 2.

Stel, er wordt een nieuwe Order opgeslagen met daarop één OrderRegel. Er wordt dan een record aangemaakt in REVISIONS waarin deze wijziging een uniek id en een timestamp krijgt. (Over de inhoud van deze tabel later meer). Tevens wordt een record aangemaakt in ORDERS_AUD en ORDER_REGELS_AUD. In deze records staan dezelfde data als in de primaire tabellen, met daarnaast een REVTYPE met waarde 0 (dit geeft aan dat het om een toevoeging gaat) en een referentie naar het record





Figuur 2: Het databaseschema van het voorbeeld.

in REVISIONS. Bij wijzigingen en verwijderingen is het proces in essentie hetzelfde; alleen het REVTYPE zal anders zijn.

Envers kan deze functionaliteit realiseren doordat Hibernate de mogelijkheid biedt om listeners te hangen aan insert-, update- en delete-events. De Envers-listeners schrijven de auditing weg. Dit betekent dat – om Envers te activeren – deze listeners dienen te worden geregistreerd in persistence.xml. Dit is een standaard stukje code dat uit de manual kan worden gekopieerd.
Om dit te laten werken moeten de extra tabellen natuurlijk wel bestaan. Net als bij standaard Hibernate kan bij Envers gebruik worden gemaakt van automatische schemacreatie. Indien schemacreatie wordt geforceerd middels de Hibernate property hbm2ddl.auto hoeft er niets specifieks te worden geregeld. Als het schema wordt gecreëerd met de Hibernate tool ant task, dan moet de EnversHibernateToolTask worden gebruikt in plaats van de standaard HibernateToolTask.

Zoeken van wijzigingen

De grootste toegevoegde waarde van Envers komt uit de mogelijkheid om niet alleen wijzigingen bij te houden, maar ook historische queries te doen op basis van de bijgehouden wijzigingen. Envers ondersteunt daarbij verschillende 'doorsneden' van de historie:

- Bij welke revisies is een gegeven entiteit toegevoegd, gewijzigd of weer verwijderd? In Enversstermen heet dit een RevisionsOfEntityQuery.
- Welke entiteiten waren er in een gegeven revisie, met welke inhoud? In Envers-termen heet dit een EntitiesAtRevisionQuery.

Het startpunt voor dergelijke queries is de AuditReader. Zo'n AuditReader kan worden verkregen via de AuditReaderFactory die daartoe een (veelal geïnjecteerde) EntityManager nodig heeft. Op deze manier werkt Envers eigenlijk als een laag om JPA heen.
De AuditReader interface biedt methods om direct de twee genoemde querytypes uit te voeren, maar kan ook worden gebruikt om een AuditQuery object te verkrijgen. Dit biedt mogelijkheden voor het doen van meer complexe queries volgens hetzelfde principe als de Criteria API in JPA2, alhoewel het niet exact dezelfde interface is. Er is geen mogelijkheid voor het doen van queries met iets als JPA query language.

In Envers worden revisies geïdentificeerd met een uniek nummer (een Number). In de praktijk is het natuurlijk vaak interessant te weten op welk tijdstip een wijziging plaatsvond, of welke revisie we moeten opzoeken om de toestand op een bepaald moment in de tijd te weten. De AuditReader interface biedt daarom ook methods om te converteren tussen tijdstip (een Date) en revisienummer.
Bij een RevisionsOfEntityQuery biedt Envers de mogelijkheid om niet alleen een lijst revisienummers, maar ook meteen de bijbehorende entiteiten en het revisietype op te vragen. Jammer daarbij is dat Envers die data dan niet-typesafe teruggeeft: het is een raw List die Object-arrays van lengte drie bevat, waarbij het eerste element de entiteit is, het tweede het revisienummer en het derde het revisietype. In de praktijk is het daarom handig een eigen (generic) 'RevisionInfo' class of iets dergelijks te maken om de output naar te converteren.

Relaties

Wanneer we niet alleen naar losse entiteiten kijken, maar ook naar entiteiten in relatie (en daar is JPA tenslotte voor bedoeld), dan ontstaan er wat complexiteiten waar je je goed van bewust moet zijn als je met Envers aan de slag gaat.
In het voorbeeld is er sprake van een one-to-many relatie tussen Order en OrderRegel. In het database schema heeft tabel ORDER_REGELS daarom de foreign key ORDER_ID verwijzend naar een record in ORDERS. In de audittabellen ligt het allemaal iets anders. Om te beginnen kan de primary key

'Versioning' of 'auditing'?

Bij Envers zie je dat de termen 'versioning' en 'auditing' door elkaar worden gebruikt. Dit heeft een historische reden. Oorspronkelijk is gestart met de term 'versioning', zoals ook in de naam Envers (entity versioning) naar voren komt. Dat bleek echter geen handige keuze, omdat in JPA het begrip 'versioning' ook al wordt gebruikt in de context van optimistic locking (met de @Version annotatie). Om deze verwarring te voorkomen, is men overgestapt op de term 'auditing'.

De extra tabellen kunnen in Envers automatisch worden aangemaakt.

Tabel ORDERS

ID	KLANTNAAM
1	Frans van Buul

Tabel ORDER_REGELS

ID	ORDER_ID	PRODUCTNAAM	AANTAL
1	1	Hibernate Search in Action	1
2	1	Design Patterns : Elements of Reusable Object-Oriented Software	3

Tabel ORDERS_AUD

ID	REV	REVTYPE	KLANTNAAM
1	1	0 (ADD)	Frans van Buul

Tabel ORDER_REGELS_AUD

ID	REV	REVTYPE	ORDER_ID	PRODUCTNAAM	AANTAL
1	1	0 (ADD)	1	Hibernate Search in Action	1
2	1	0 (ADD)	1	Design Patterns : Elements of Reusable Object-Oriented Software	1
2	2	1 (MOD)	1	Design Patterns : Elements of Reusable Object-Oriented Software	3

Tabel REVISIONS

ID	REVTIME	USERID
1	2010-01-30 20:56:03.321	fvb
2	2010-01-30 20:58:38.518	fvb

Figuur 3: De gevulde database.

van een reguliere tabel nooit tevens de primary key van de audittabel zijn, omdat er meerdere versies (met dezelfde primary key) moeten worden opgeslagen. De primary key van de audittabellen is daarom altijd de combinatie van de oorspronkelijke primary key met de revisie.
Records in audittabellen kunnen weliswaar verwijzen naar elkaar, maar er is geen sprake van harde foreign keys. Het waarom hiervan is het makkelijkst in te zien met een voorbeeld. Stel, er wordt in eerste instantie een Order aangemaakt met twee OrderRegels. Daarna vindt een wijziging plaats van de tweede OrderRegel; veld 'aantal' wordt verhoogd van 1 naar 3. De database is dan ongeveer gevuld zoals in Figuur 3.

Dit betekent dat het ophalen van entiteiten voor revisie 2 niet zo simpel is als het beperken van de audit tabellen tot records met revisie 2. Immers, alleen wijzigingen worden weggeschreven. De gewijzigde OrderRegel met id 2 en revisie 2 is nog steeds gerelateerd aan de Order met id 1 en revisie 1. Om de juiste entities te zoeken, moet Envers daarom het volgende principe volgen: de inhoud van een entiteit in een gegeven revisie wordt gegeven door het record in de audittabel met het hoogste revisienummer kleiner of gelijk aan het gevraagde revisienummer.

Dit principe was bij het ontwerp van Envers onvermijdelijk. Het alternatief zou zijn om bij een wijziging in een OrderRegel ook een nieuwe revisie van Order op te slaan. Maar in een typisch relationeel model zijn alle entiteiten uiteindelijk op enige manier aan elkaar verbonden; de hoeveelheid data die per wijziging zou moeten worden opgeslagen, zou exploderen.
Desondanks heeft dit principe wel de nodige consequenties waar de ontwikkelaar rekening mee moet houden.

Ten eerste: allerlei annotaties rondom eager/lazy fetching en fetch methods als join en subselect, worden volledig genegeerd bij Envers queries. Envers fetching is altijd lazy, en wordt altijd uitgevoerd met aparte select statements. Om de OrderRegels bij een Order op te halen, voert Envers de volgende SQL query uit:

```
select
   orderregel0_.id as col_0_0_,
   orderregel0_.REV as col_0_1_
from
   ORDER_REGELS_AUD orderregel0_
where
   orderregel0_.ORDER_ID=?
   and orderregel0_.REV=(
        select
           max(orderregel1_.REV)
        from
           ORDER_REGELS_AUD orderregel1_
        where
           orderregel1_.REV<=?
           and orderregel0_.id=orderregel1_.id
        )
   and orderregel0_.REVTYPE<>?
```

Het moge tevens duidelijk zijn dat dit type query nooit de performance gaat halen van een reguliere JPA query.

Ten tweede: er moet soms rekening gehouden worden met een verschil tussen wat Envers als een wijziging ziet, en wat meer vanuit een gebruikersperspectief een wijziging is. In het voorbeeld werd de tweede regel van een Order gewijzigd. Vanuit een gebruikersperspectief betekent dit dat de Order is gewijzigd. Maar als je in Envers een RevisionsOfEntityQuery doet voor de Order, zul je geen revisie zien die correspondeert met de wijziging in OrderRegel! Overigens ziet Envers (met de standaardinstellingen) het toevoegen of verwijderen van een element in een one-to-many relatie wél als een wijziging in de bovenliggende entiteit.

Extra revisiegegevens

In het voorbeeld hebben we de Revision entiteit en de bijbehorende REVISIONS tabel zonder veel toelichting geïntroduceerd. In deze tabel schrijft Envers een record weg voor iedere wijziging. Standaard bevat deze data niets meer dan een uniek nummer en een timestamp in de vorm van een long waarde.

Envers is geen JPA-generieke oplossing, maar een Hibernate-specifieke oplossing.

Een zeer interessante feature van Envers is dat we zelf de revision-entiteit mogen bepalen. Daarbij kan de timestamp veranderd worden van een long in een Date (wat het makkelijker maakt om in de database de timestamps te bekijken), maar kunnen er ook velden aan worden toegevoegd. Het definiëren van een eigen revision-entiteit gaat met behulp van een paar annotaties. De basis is een standaard JPA entity. Met *@RevisionEntity* kun je aangeven dat het een revision-entiteit is. De annotaties *@RevisionNumber* en *@RevisionTimestamp* geven aan welke attributen de basisgegevens voor de revision bevatten. Als parameter van de *@RevisionEntity* annotatie kan een listener worden opgegeven die wordt aangeroepen bij het maken van een nieuwe revisie; deze listener, die de Envers RevisionListener interface moet implementeren, kan de aanvullende velden vullen.

Een zeer voor de hand liggende toepassing hiervoor is het wegschrijven van de gebruiker die verantwoordelijk was voor de wijziging. Dit is bij auditing op databaseniveau erg lastig, omdat Java-applicaties meestal onder één constante user naar de database gaan, en de database dus niet weet welke eindgebruiker bezig is.

Een probleem dat daarbij wel moet worden opgelost, is hoe de revisionlistener kan bepalen wie de actieve gebruiker is. De beste oplossing daarvoor hangt af van het gebruikte applicatieframework. In een eenvoudige Java SE applicatie zou dit kunnen worden bijgehouden middels een globaal ThreadLocal object. In een Java EE context ligt het voor de hand hiervoor de TransactionSynchronizationRegistry te gebruiken, die door de revisionlistener middels een JNDI-lookup kan worden verkregen.

Conclusie

Envers biedt auditing faciliteiten op JPA-niveau. Ten opzichte van eigen auditing code op applicatieniveau heeft dit als voordeel dat er nauwelijks eigen code hoeft te worden geschreven. Ten opzichte van auditing faciliteiten op database-niveau heeft dit als voordeel dat de oplossing databaseonafhankelijk is, dat schemacreatie blijft werken, dat er historische queries kunnen worden gedaan, en dat gegevens zoals de verantwoordelijke gebruiker kunnen worden toegevoegd.

De architectuur van Envers heeft ook enkele beperkingen. Allereerst is dit geen JPA-generieke oplossing, maar een Hibernate-specifieke oplossing. In situaties waarin een andere JPA provider wordt gebruikt dan Hibernate (b.v. EclipseLink), kan Envers niet worden ingezet. Daarnaast 'ziet' Envers alleen maar wijzigingen die via JPA worden doorgevoerd. In gevallen waarin er meerdere bronnen van wijzigingen in de database zijn dan alleen de Java-applicatie, is de toepasbaarheid van Envers beperkt.

In een groot aantal gevallen zullen deze beperkingen echter geen bezwaar vormen, en kan met Envers op een eenvoudige wijze worden voldaan aan functionele eisen op het gebied van auditing! «

Welke versie van Envers?

Als je Envers gebruikt, moet de versie aansluiten bij de versie van Hibernate. Sinds Hibernate 3.5 zijn Hibernate versienummers gemakkelijk: gelijke nummers zijn compatible. Voor Hibernate-3.5.6-Final heb je dus ook Hibernate Envers 3.5.6-Final nodig.

Voor oudere versies is het even puzzelen. Op een JBoss 5.1 server worden bijvoorbeeld Hibernate EntityManager 3.4.0.GA en Hibernate Core 3.3.1.GA gebruikt. Voor die versie was Envers nog geen Hibernate module, maar heette het JBoss Envers. De vereiste versie is 1.2.2.GA-hibernate-3.3.

Envers en de bijbehorende documentatie zijn te vinden via http://www.hibernate.org/. De oudere versie staat op http://www.jboss.org/envers