

# ¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.  
 Ese apoyo que siempre quiso tener...

## 1. Desarrollo de componentes y proyectos a medida



## 2. Auditoría de código y recomendaciones de mejora

## 3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



## 4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,  
 HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)  
 Gestor de contenidos (Alfresco)  
 Aplicaciones híbridas

Tareas programadas (Quartz)  
 Gestor documental (Alfresco)  
 Inversión de control (Spring)

Control de autenticación y  
 acceso (Spring Security)  
 UDDI  
 Web Services  
 Rest Services  
 Social SSO  
 SSO (Cas)

JPA-Hibernate, MyBatis  
 Motor de búsqueda empresarial (Solr)  
 ETL (Talend)

Dirección de Proyectos Informáticos.  
 Metodologías ágiles  
 Patrones de diseño  
 TDD

BPM (jBPM o Bonita)  
 Generación de informes (JasperReport)  
 ESB (Open ESB)



E-mail:   
Contraseña:

Entrar

Deseo registrarme  
He olvidado mis datos de acceso

- [Inicio](#) [Quiénes somos](#) [Tutoriales](#) [Formación](#) [Comparador de salarios](#) [Nuestro libro](#) [Charlas](#) [Más](#)

Estás en: [Inicio](#) [Tutoriales](#) [Mapeo de Procedimientos Almacenados con Hibernate](#)

 <p><b>DESARROLLADO POR:</b> Juan Alonso Ramos</p>	<p>Consultor tecnológico de desarrollo de proyectos informáticos.</p> <p>Ingeniero Técnico en Informática de Gestión e Ingeniero en Informática, especialidad en Ingeniería del Software</p> <p>Puedes encontrarme en Autentia: Ofrecemos de servicios soporte a desarrollo, factoría y formación</p> <p>Somos expertos en Java/J2EE</p>
--	--

Catálogo de servicios Autentia

- [Anuncios Google](#) [Java Tutorial Classes](#) [Tutorial](#) [Oracle PL SQL Tutorial](#) [Hibernate](#)

Fecha de publicación del tutorial: 2009-02-26



Share |

Regístrate para votar

## Mapeo de Procedimientos Almacenados con Hibernate

### Índice de contenidos.

- 1. Introducción
- 2. Entorno
- 3. Procedimiento Almacenado
- 4. Mapeo de Hibernate
- 5. Llamada a la NamedNativeQuery
- 6. Conclusiones

### 1. Introducción

En este tutorial vamos a ver la forma de trabajar con Hibernate para que podamos llamar a los procedimientos almacenados de nuestra base de datos. En muchos desarrollos es común ver como tienen toda o parte de la lógica de negocio en la base de datos. En estos casos, debido a que nos vamos a replicar esta lógica de negocio en el código fuente de la aplicación para utilizarlo desde la capa de negocio, podemos invocarlo mediante queries nativas y el resultado de las mismas a entidades para que sea más sencillo trabajar con los datos.

La forma de mapear las queries nativas está disponible tanto vía anotaciones como mediante el fichero de mapeo hbm.xml. En este tutorial usaremos anotaciones. Más info aquí.

Para la base de datos utilizaré PostgreSQL.

Para hacer las pruebas me basaré en el código fuente del siguiente [tutorial](#).

El código fuente del ejemplo lo puedes descargar de [aquí](#).

### 2. Entorno

- MacBook Pro 15' (2.4 GHz Intel Core i5, 4GB DDR3 SDRAM).
- Sistema Operativo: Mac OS X Snow Leopard 10.6.4
- JDK 1.6.0\_20
- Hibernate Core 3.3.2.GA
- PostgreSQL 9.0.2
- JUnit 4.7

### 3. Procedimiento Almacenado

Lo primero que haremos será crearnos un procedimiento almacenado en PostgreSQL para posteriormente mapearlo con Hibernate. La lógica del procedimiento es lo de menos, en este caso el procedimiento **findByNumber** se encarga de devolver un registro con los datos del futbolista al cual corresponde el número recibido por parámetro.

```
01 CREATE OR REPLACE FUNCTION findByNumber(integer) RETURNS record AS $body$
02 DECLARE
03     numberToFind ALIAS FOR $1;
04     footballer record;
05
06 BEGIN
07     select id, name, lastname, number INTO footballer FROM footballer WHERE number = numberToFind;
08
09     RETURN footballer;
10 END;
11 $body$ LANGUAGE plpgsql;
```

### 4. Mapeo de Hibernate

Una vez que tenemos el procedimiento almacenado en la base de datos vamos a mapearlo a través de una query nativa. Las queries nativas se utilizan para escribir en sql estándar (en lugar de HQL) una consulta que pedimos a Hibernate que lance. Mediante esta query nativa llamamos al procedimiento almacenado y su resultado se guardará en la entidad Footballer.

```
01 @Entity
02 @NamedNativeQuery(name = "Footballer.findByNumber", query = "select * from findByNumber(?) as (id int, name
03 varchar, lastname varchar, number int);", resultClass = Footballer.class)
04 public class Footballer {
05
06     @Id
07     @GeneratedValue
08     private Integer id;
09
10     private String name;
11
12     private String lastname;
13
14     private int number;
15
16     ...
17     // Se omiten los getters y setters
```

A través de la anotación `@NamedNativeQuery` indicamos la query nativa llamada `Footballer.findByNumber`. La query se especifica mediante el atributo `query` donde hacemos la llamada al procedimiento almacenado que definimos anteriormente. Debido a que lo que nos devuelve la base de datos es un registro debemos sacar los datos para posteriormente ser mapeados a través del 'select ... as' indicando su nombre y tipo como se muestra en la consulta anterior. Por último, a través del parámetro `resultClass` indicamos la clase donde mapear los datos, en este caso la clase `Footballer`. No es necesario indicar los atributos de la clase si se llaman igual que los que devuelve la consulta que llama al procedimiento. Si fueran diferentes se lo debemos indicar a través de un `resultSetMapping` mapeando tanto la entidad destino de los datos como sus atributos. Este sería el ejemplo de ese caso:

```
01 @Entity
02 @NamedNativeQuery(name = "Footballer.findByNumber", query = "select * from findByNumber(?) as (identificador int,
```

### Últimas Noticias

- [Pequeño coding dojo con Carlos Ble en las oficinas de Autentia.](#)
- [Disponible gratis, Autentia Comic para el iPhone y iPad,](#)
- [Comentando #AID2010. Agil Industrial Day 30 Nov 2010](#)
- [Autentia Head Hunting - ¡¡Primeros contratos!!](#)
- [XIII Charla Autentia - AOS y TDD](#)



### Últimos Tutoriales

- [Autoescaneo de entidades de Hibernate con Spring](#)
- [Cómo listar una entidad en wuija con prefiltrado](#)
- [jQuery: Paginación](#)
- [Girillo TTS: Una aplicación Android para evitar el uso del móvil durante la conducción](#)
- [Ejemplo básico de Spring MVC Portlet](#)

### Últimos Tutoriales del Autor

- [Autoescaneo de entidades de Hibernate con Spring](#)
- [DataTable con paginación en base de datos con Primefaces](#)
- [Generación de Informes con JasperReports en PHP](#)
- [Múltiples datasources en JasperReports](#)
- [JCaptcha - Generación de Captchas en Java](#)

### Síguenos a través de:



### Últimas ofertas de empleo

- 2010-10-11 [Comercial - Ventas - SEVILLA.](#)
- 2010-08-30 [Otras - Electricidad - BARCELONA.](#)
- 2010-08-24 [Otras Sin catalogar - LUGO.](#)

```

03 nombre varchar, apellidos varchar, numero int);", resultSetMapping = "mapping")
04 @SqlResultSetMapping(name = "mapping", entities = {
05   @EntityResult(entityClass = Footballer.class, fields = {
06     @FieldResult(name = "id", column = "identificador"),
07     @FieldResult(name = "name", column = "nombre"),
08     @FieldResult(name = "lastname", column = "apellidos"),
09     @FieldResult(name = "number", column = "numero") }) })
10
11   @Id
12   @GeneratedValue
13   private Integer id;
14
15   private String name;
16
17   private String lastname;
18
19   private int number;
20
21   ...
22   // Se omiten los getters y setters

```

De esta manera le decimos a Hibernate cómo debe mapear los datos en los atributos de la entidad. Esta forma es menos recomendable que la primera ya que implica mayor configuración, recordar el patrón Convention over Configuration. Si decidimos utilizarla debemos indicar el `resultSetMapping` que indicará la forma de mapear los datos devueltos por la consulta. Para ello definimos un "mapping" mediante la anotación `SqlResultSetMapping` donde configuramos la entidad destino de los datos "Footballer.class" y los campos dentro de la entidad donde recoger los datos de la query a través de los `FieldResult`.

## 5. Llamada a la NamedNativeQuery

Por último ya únicamente falta la llamada al procedimiento almacenado, para ello nos creamos un método en nuestro Dao llamado "findByNamedQuery". Este método recibe el nombre de la namedQuery, parámetros para indicar el primer registro a sacar en la consulta y el máximo de registros y por último los filtros de la consulta.

```

01 public <T> List<T> findByNamedQuery(String namedQuery, int firstResult, int maxResults, Object... values) {
02
03   final Session session = getHibernateTemplate().getSessionFactory().getCurrentSession();
04   final Query query = session.getNamedQuery(namedQuery);
05
06   for (int i = 0; i < values.length; i++) {
07     query.setParameter(i, values[i]);
08   }
09
10   if (firstResult > 0) {
11     query.setFirstResult(firstResult);
12   }
13
14   if (maxResults > 0) {
15     query.setMaxResults(maxResults);
16   }
17
18   return query.list();
19 }

```

Para comprobar que todo está bien lo probamos mediante un test de JUnit. Antes de ejecutarse el test Spring realizará la llamada al método init del bean `AddSampleData` que proveerá de datos a la base de datos.

```

01 package com.autentia.tutoriales.dao.service.impl;
02
03 // imports
04
05 @Service
06 public class AddSampleData extends HibernateDaoSupport {
07
08   private Dao dao;
09
10   @Autowired
11   public AddSampleData(Dao dao, SessionFactory factory) {
12     super.setSessionFactory(factory);
13     this.dao = dao;
14   }
15
16   @SuppressWarnings("unused")
17   @PostConstruct
18   private void init() {
19     final List<Footballer> list = new ArrayList<Footballer>();
20
21     final Footballer iker = new Footballer();
22     iker.setName("Iker");
23     iker.setLastname("Casillas");
24     iker.setNumber(1);
25     list.add(iker);
26
27     final Footballer pepe = new Footballer();
28     pepe.setName("Pepe");
29     pepe.setLastname("Lima");
30     pepe.setNumber(3);
31     list.add(pepe);
32
33     final Footballer sergio = new Footballer();
34     sergio.setName("Sergio");
35     sergio.setLastname("Ramos");
36     sergio.setNumber(4);
37     list.add(sergio);
38
39     // ...
40
41     dao.saveOrUpdate(list);
42   }

```

El test de JUnit busca un futbolista por su número de camiseta llamando al procedimiento almacenado "Footballer.findByNumber" y comprueba que se recibe en el listado un futbolista que cumple los criterios de la búsqueda. Se le pasa al método `findByNamedQuery` los valores `firstResult = -1` y `maxResult = -1` ya que no son relevantes en esta consulta.

```

01 package com.autentia.tutoriales;
02
03 // imports
04
05 @RunWith(SpringJUnit4ClassRunner.class)
06 @ContextConfiguration(locations = { "classpath:applicationContext.xml" })
07 public class NativeQueryTest {
08
09   @Resource
10   private Dao dao;
11
12   private static final int IKER_CASILLAS_NUMBER = 1;
13
14   @Test
15   public void testFindByNumberByStoreProcedure() {
16     final List<Footballer> footballerList = dao.findByNamedQuery("Footballer.findByNumber", -1, -1,
17       IKER_CASILLAS_NUMBER);
18
19     Assert.assertEquals("Deberia haber un futbolista ", 1, footballerList.size());
20
21     final Footballer footballer = footballerList.get(0);
22
23     Assert.assertEquals("Iker", footballer.getName());
24     Assert.assertEquals("Casillas", footballer.getLastname());
25     Assert.assertEquals(IKER_CASILLAS_NUMBER, footballer.getNumber());
26   }

```

Para completar la prueba de concepto vamos a crear otro procedimiento almacenado que se encargará de sumar dos números y devolver el resultado de la suma. Lógicamente la funcionalidad del procedimiento es lo de menos, lo que nos interesa en este caso es el resultado de una operación que no formará parte de una entidad sino que será un escalar por lo que los mapeos son diferentes.

Lo primero será crear el procedimiento almacenado encargado de realizar la suma:

```

01 CREATE OR REPLACE FUNCTION suma(operando1 integer, operando2 integer) RETURNS int AS $body$
02 DECLARE
03     operando1 ALIAS FOR $1;
04     operando2 ALIAS FOR $2;
05     resultado int;
06
07 BEGIN
08     resultado = operando1 + operando2;
09
10     RETURN resultado;
11 END;
12 $body$ LANGUAGE plpgsql;

```

Creamos una nueva **NamedNativeQuery** en la entidad **Footballer**. Se mete en esta entidad por simplificar el ejemplo aunque no tenga nada que ver ;-)

```

1 @Entity
2 @NamedNativeQueries({
3     @NamedNativeQuery(name = "Footballer.findByNumber", query = "select * from findByNumber(?) as (id int,
4     name varchar, lastname varchar, number int);", resultClass = Footballer.class),
5     @NamedNativeQuery(name = "Footballer.SumaDeEnteros", query = "select suma(?,?) as resultado",
6     resultSetMapping = "scalar" ) })
7 @SqlResultSetMapping(name = "scalar", columns = @ColumnResult(name = "resultado"))
8 public class Footballer {
9     // ....
10 }

```

Mediante la query nativa "Footballer.SumaDeEnteros" llamamos al procedimiento almacenado "suma" que recibe dos enteros y devuelve el resultado. Este resultado será un valor escalar y la forma de indicarle a Hibernate que trate este valor es mediante el **SqlResultSetMapping** donde se define el nombre que llevará el dato en el ResultSet de la consulta.

Para probar esta nueva funcionalidad añadimos un nuevo test de JUnit donde realizamos la llamada a la NativeNamedQuery pasándole los valores 2 y 3 y comprobamos el valor de la suma.

```

1 @Test
2 public void testSumaByStoreProcedure() {
3     final List<Integer> list = dao.findByNameQuery("Footballer.SumaDeEnteros", -1, -1, 2, 3);
4     Assert.assertEquals(Integer.valueOf(5), list.get(0));
5 }

```

## 6. Conclusiones

Muchas veces hemos escuchado que una aplicación o módulo funcional no se puede migrar a Hibernate (o Java) debido a que hay mucha lógica de negocio en procedimientos almacenados de la base de datos. Como hemos podido ver no es excusa tener estos procedimientos ya que Hibernate es capaz de llamar a esta lógica y mapear el resultado por lo que nos podemos beneficiar de las grandes ventajas que nos proporciona Hibernate para trabajar con el modelo de datos.

Espero que te haya servido de ayuda.

Un saludo. Juan.

¡Anímate y coméntanos lo que pienses sobre este **TUTORIAL**!

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

[Enviar comentario](#)

(Sólo para usuarios registrados)

» [Regístrate](#) y accede a esta y otras ventajas «

## COMENTARIOS



Esta obra está licenciada bajo licencia [Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)