MANUAL DE HIBERNATE

ÍNDEX DE CONTINGUTS

INTRODUCCIÓ

Aquest manual no està destinat a ser una manual complet de JPA i, conseqüentment, Hibernate, sinó una guia indicativa que hauria de servir a l'estudiant per poder resoldre la majoria de problemes que es pot trobar al iniciar-se en aquestes tecnologies. Per tant, és possible, i altament probable, que sorgeixin dubtes que aquest manual no resol així que la recomanació de l'autor és visitar la documentació completa de Hibernate i revisar l'api de JPA per a tal de resoldre'ls.

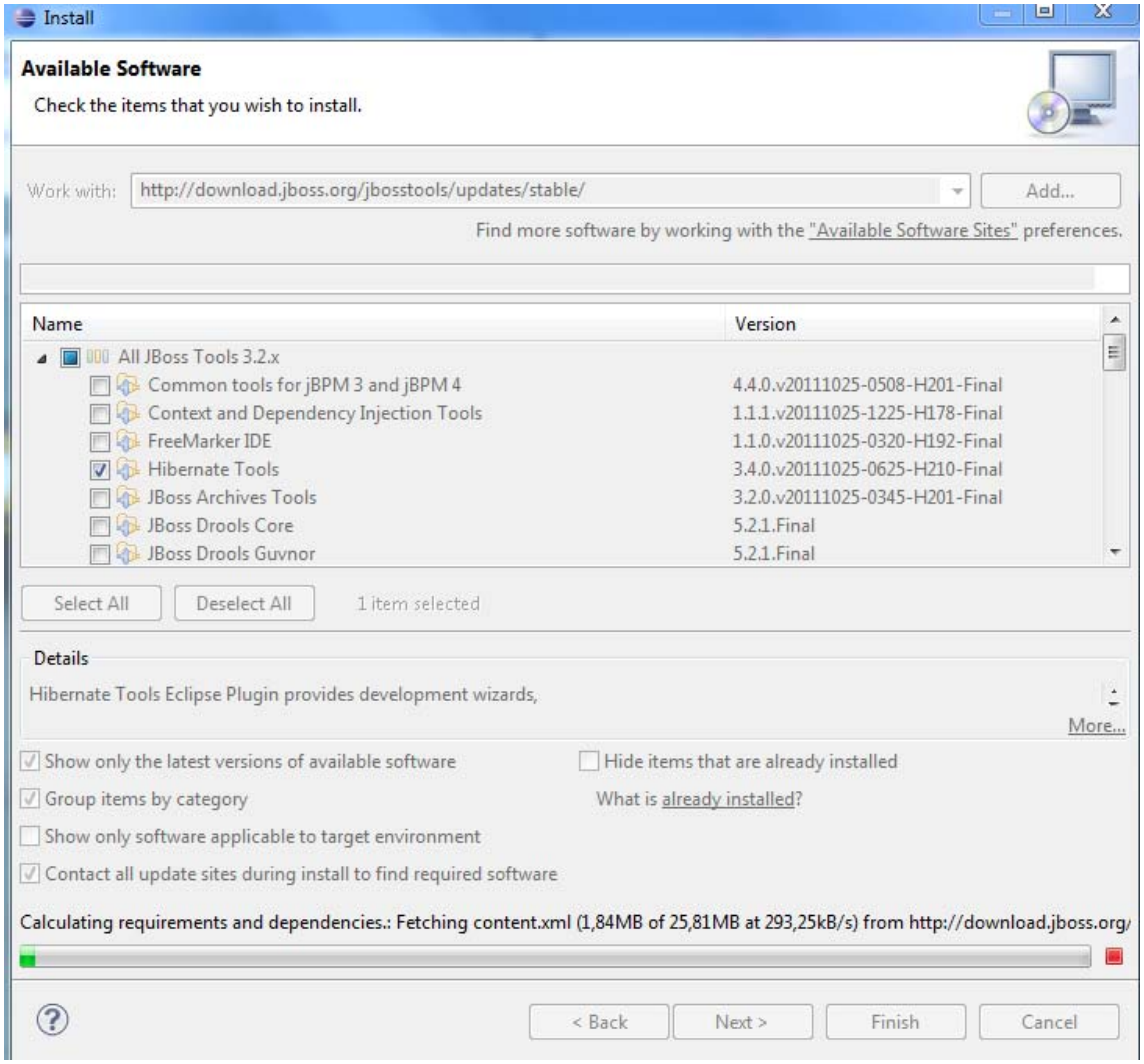
També es proporciona una font alternativa per a la introducció de l'estudiant a JPA i Hibernate en forma d'un conjunt de 18 vídeos educatius, l'adreça on es poden trobar és: http://www.youtube.com/watch?v=GINvxAaXDbY&list=ULTanXgmG6ZuU.

HIBERNATE PER A ECLIPSE

A continuació s'explicarà breument el procediment d'instal·lació de Hibernate per a Eclipse i la configuració d'aquest. A més, al final s'inclouen també solucions a possibles problemes que poden sorgir en el seu ús. La versió utilitzada per a aquest manual és la 3.7.4 Indigo amb les llibreries de Java, es pot obtenir d'aquí : http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/indigosr2.

INSTAL·LACIÓ

Un cop tinguem el Eclipse descarregat i funcionant, haurem d'instal·lar l'extensió de *Hibernate Tools* des de http://download.jboss.org/jbosstools/updates/stable, a continuació es mostra una imatge del que s'hauria d'instal·lar.



En el procés d'instal·lació del software és possible que ens aparegui un avís avisant que el software no està firmat. És segur acceptar-ho i continuar la instal·lació.

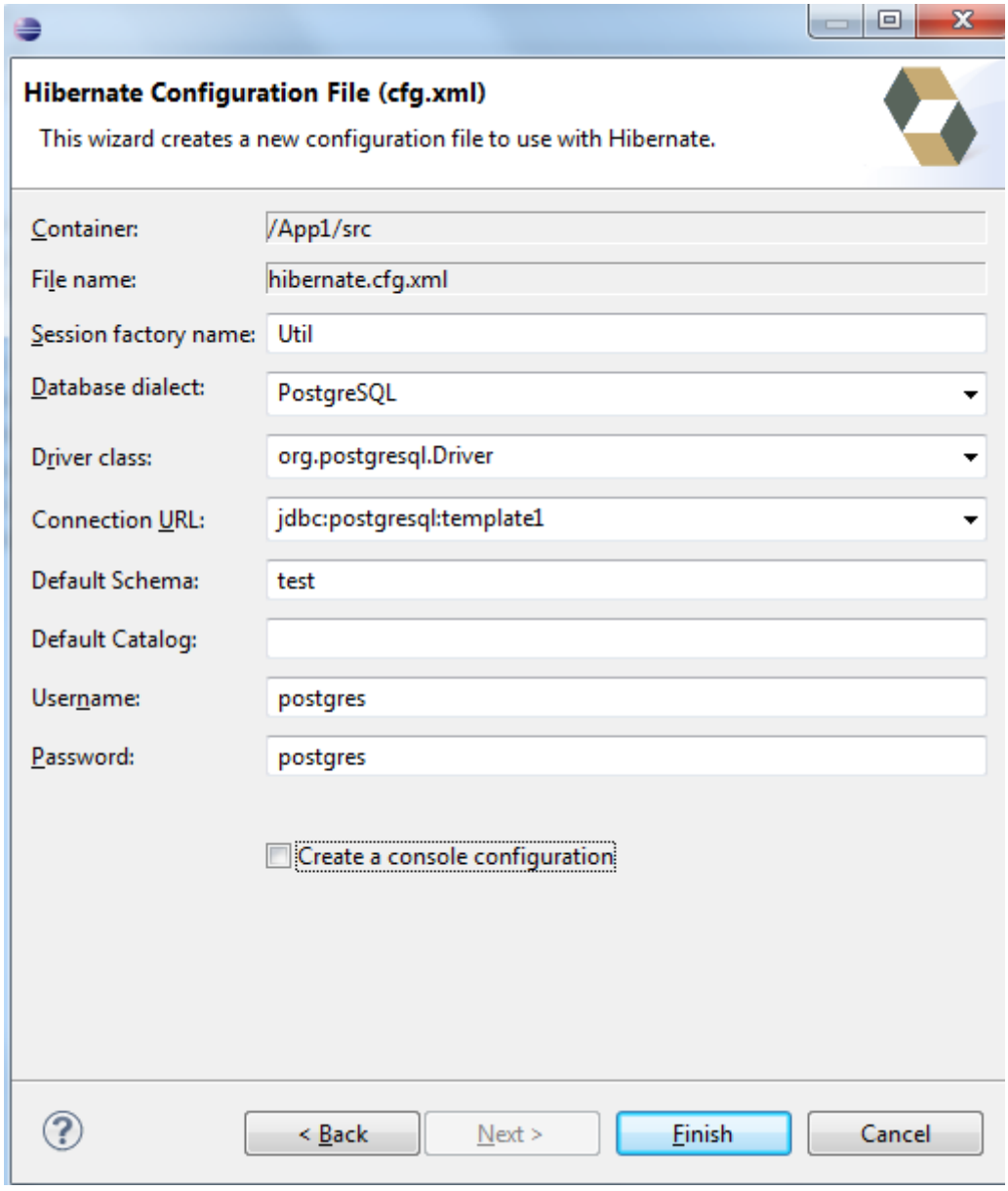
Un cop instal·lat i reiniciat ja podrem utilitzar el Hibernate per a Eclipse.

UTILITZACIÓ

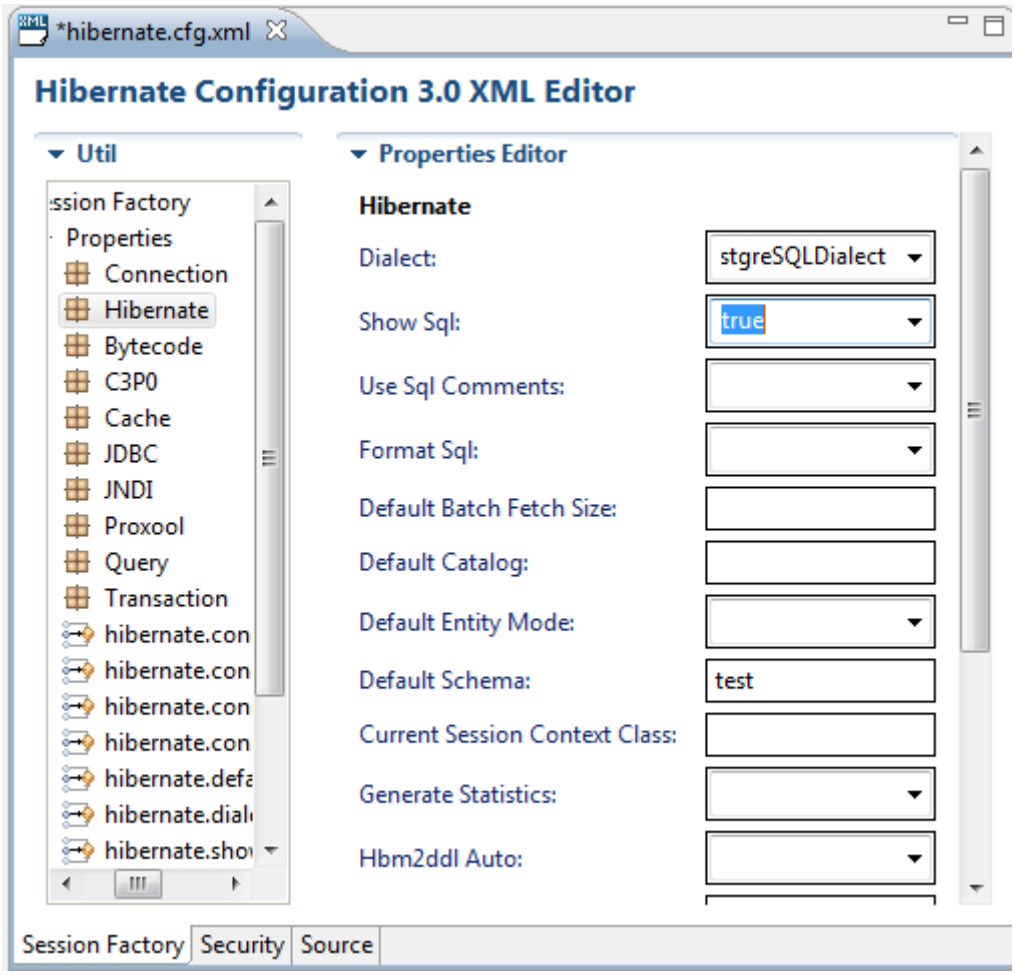
Un cop creat un projecte qualssevol, anem a les propietats d'aquest. Allà veurem que hi ha una nova pestanya anomenada *Ajustaments de Hibernate*. En aquesta pestanya n'habilitem el suport i li donem a aplicar. Ara ja podem crear fitxers de Hibernate, anant a *Fitxer => Nou => Altres* i seleccionant el desitjat de la carpeta Hibernate.

ARXIU DE CONFIGURACIÓ DE HIBERNATE

El procés de configuració és lleugerament diferent de l'utilitzat a NetBeans. En primer lloc creem un nou fitxer utilitzant la plantilla *Hibernate Configuration File (cfg.XML)* i allà completem els camps seguint un ordre descendent (això és, perquè al seleccionar el dialecte ens permetrà seleccionar el controlador).

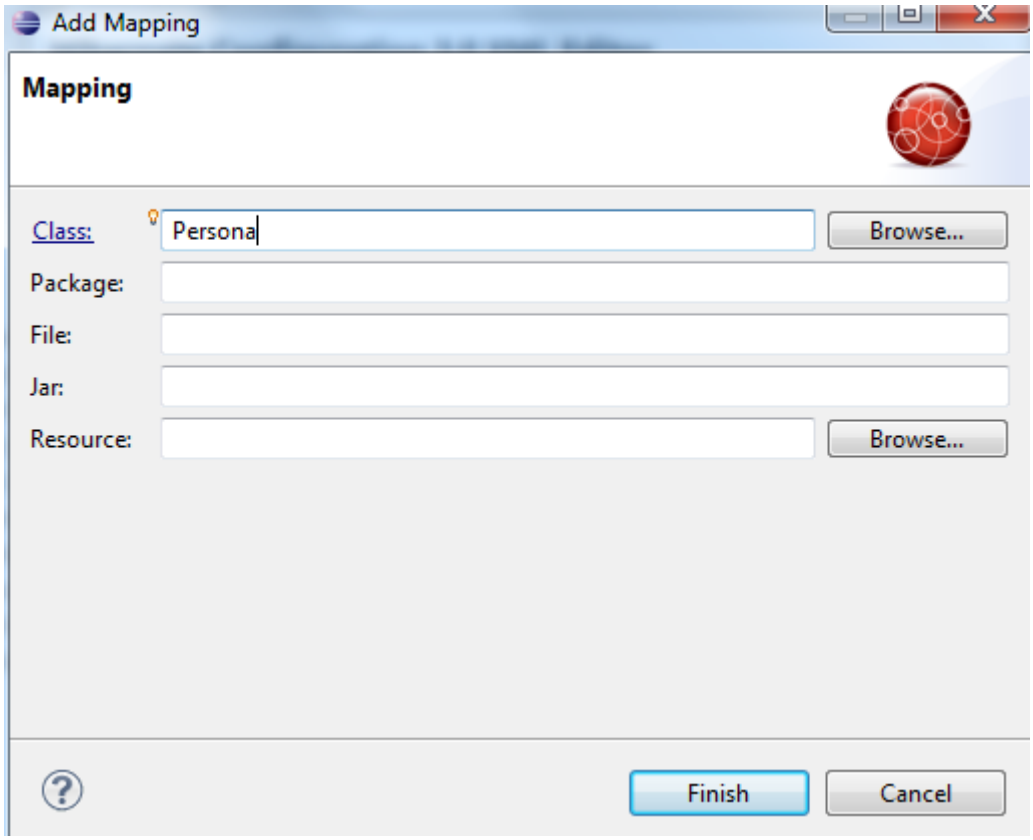


Un cop creat l'arxiu se'ns obrirà automàticament l'editor. A diferència del NetBeans, aquí haurem d'afegir les propietats manualment tot obrint el menú contextual a *Propietats* i seguint amb *Nova Propietat*. Alternativament també es pot fer com en la imatge a continuació:



MAPEIGS

Els mapeigs s'afegeixen en el camp *Mapeigs* del fitxer de configuració, clicant a afegir. A continuació podem veure un exemple de configuració:



ALTRES

Un pas opcional és el de crear un arxiu de configuració de consola, això és, poder modificar la base de dades des del propi Hibernate. Podeu consultar a la següent direcció web com fer-ho: http://docs.redhat.com/docs/en-US/JBoss_Developer_Studio/4.0/html/Hibernate_Tools_Reference_Guide/console_conf.html.

HIBERNATE PER A NETBEANS

Aquest manual està fet utilitzant NetBeans 7.0.1 (http:/netbeans.org/downloads/7.0.1/start.html?%20lang=en&option=javase) en la seva versió anglesa, en altres versions pot canviar l'apartat de configuració. Està basat, en part, en el manual oficial a: http://NetBeans.org/kb/docs/Java/Hibernate-Java-se.html.

En primer lloc creem un projecte qualssevol, per exemple unaJava Desktop Applicationamb swing. A aquest projecte afegim la llibreria *Hibernate JPA* (això és, clic dret sobre el projecte, seleccionar *Llibreries* i posteriorment clicar a *Afegir Llibreria*). Aquesta llibreria incorpora, a més, plantilles d'arxius de Hibernate.

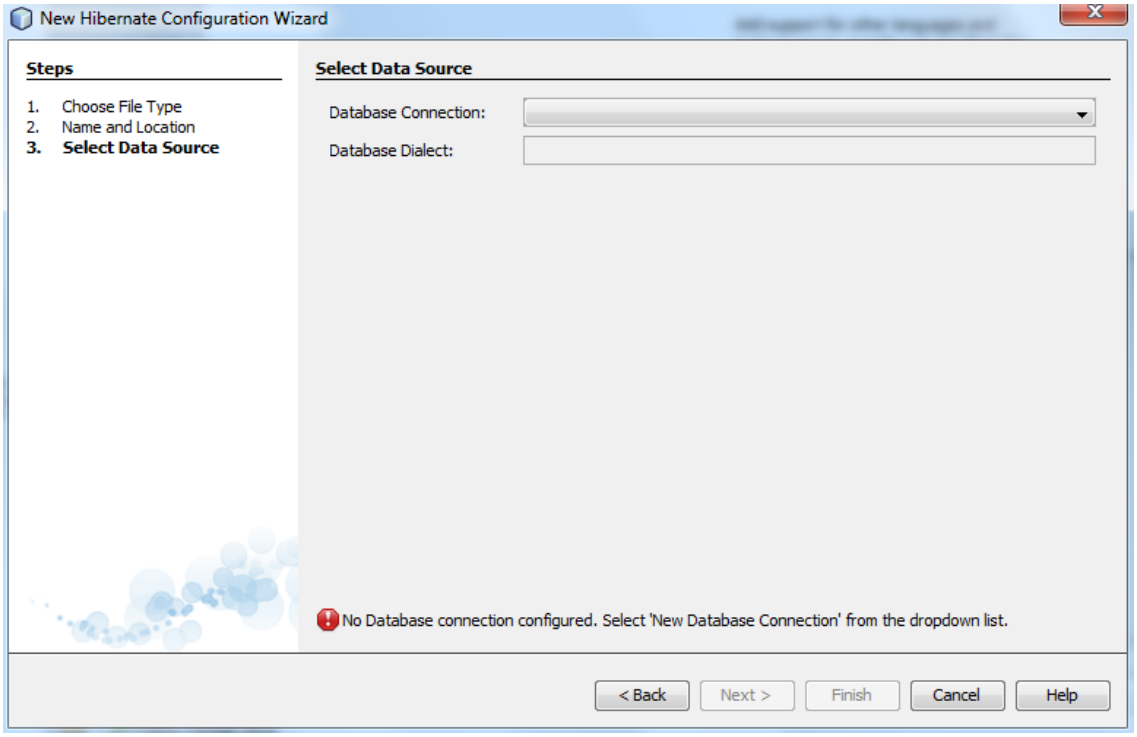
ARXIU DE CONFIGURACIÓ DE HIBERNATE

L'arxiu de configuració de Hibernate conté tota la informació relacionada amb la connexió amb la base de dades, els arxius que Hibernate haurà de processar i d'altres. Al ser un fitxer XML, anomenat *Hibernate.cfg.XML*, aquest es pot crear manualment i definir-ne les claus segons les opcions desitjades. Ara bé, NetBeans ja ens ofereix un assistent per a crear-lo, de forma que acaba essent molt més senzill.

- Clic dret enSource Packagesa la finestra de projectes i seleccionarNewi, posteriorment,Other'.
- Seleccionar *Hibernate Configuration Wizard* de la categoria *Hibernate*, prémer següent.
- Deixar el nom i la localització per defecte, ja que el fitxer ha d'estar a la carpetasrcper a que tot funcioni correctament, continuar.

CONNEXIÓ A LA BASE DE DADES

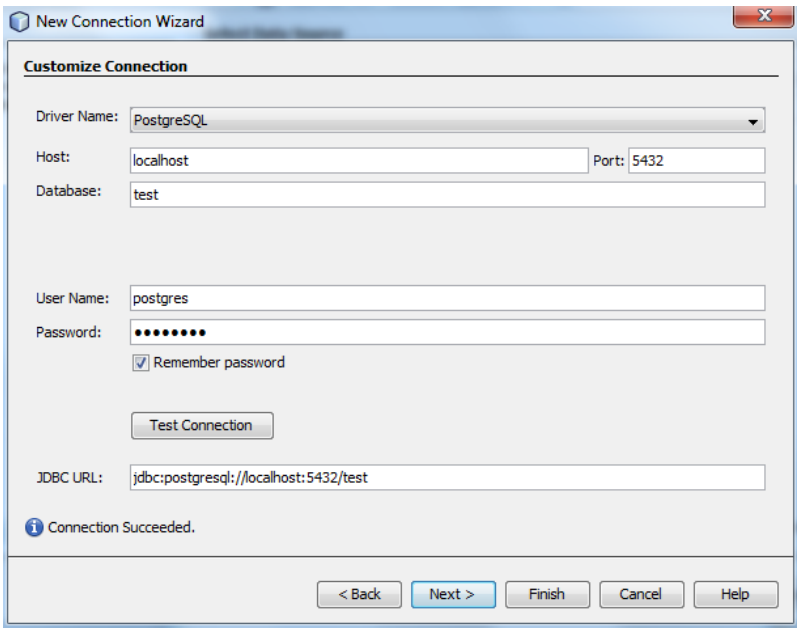
S'ha de configurar la connexió a la base de dades perquè Hibernate funcioni correctament. Si encara no s'ha configurat la connexió a la base de dades no hi haurà cap connexió (o les que hi han no són les que volem) marcada, així que fem clic sobre el menú desplegable al costat de *Database Connection* i seleccionem *New Database connection*.



Els passos descrits a continuació són per a una connexió amb PostrgreSQL tot i que també serveixen per a qualssevol altre base de dades.

Si en el menú desplegable *Controlador* hi apareix PostrgreSQL es pot obviar aquest paràgraf, sinó a *Controlador* seleccionem *Nou Controlador*. Aquí premem *Afegir* i cerquem el fitxer del controlador (normalment, un fitxer *.jar*). Un cop seleccionat el fitxer del controlador els altres camps s'haurien d'emplenar automàticament. Si no és així assegurar-se que el controlador és correcte i, si tot i així no s'aconsegueix, emplenar els camps manualment: Al camp *Classe del Controlador* s'hi haurà de posar la classe principal, que és la que al seu torn implementa *java.sql.Controlador*.

Un cop ja marcat el controlador li continuem a la plana següent i emplenem els camps a l'estil com apareixen en la figura a continuació.



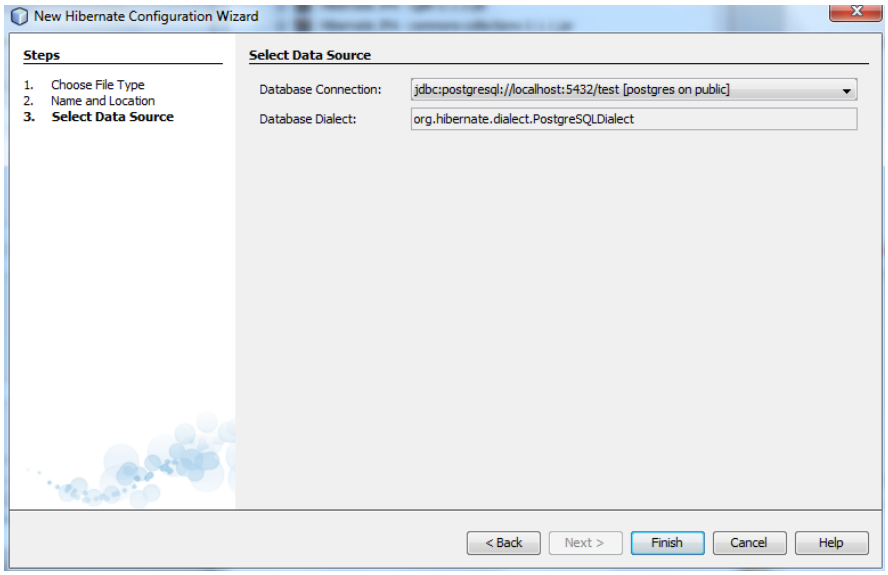
Un cop fet això podem comprovar que funciona mitjançant *Provar Connexió*, que ens hauria de mostrar *Connexió realitzada amb èxit*, tal i com mostra la figura.

Cliquem Següent i seleccionem el esquema pertinent. Es recomana no utilitzar l'esquema per defecte si la base de dades s'utilitza per a altres finalitats.

Finalment seleccionem Finalitzar, la pantalla de l'assistent ens hauria de quedar semblant al exemple mostrat a continuació. Un cop revisat que tot estigui correcte ja podem tancar l'assistent.

PERSONALITZAR LA CONFIGURACIÓ DE HIBERNATE

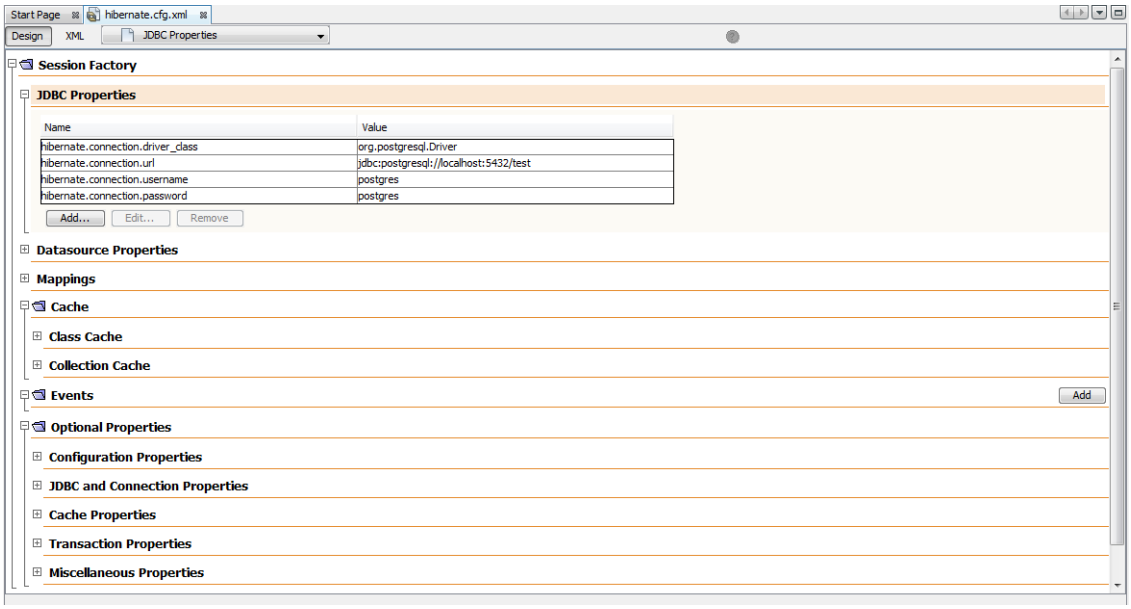
Els passos següents són optatius però altament recomanats ja que ajuden en el desenvolupament d'aplicacions per a Hibernate. Aquí s'explicarà com fer-ho amb el dissenyador de NetBeans, però modificant el fitxer XML es poden obtenir els mateixos resultats.



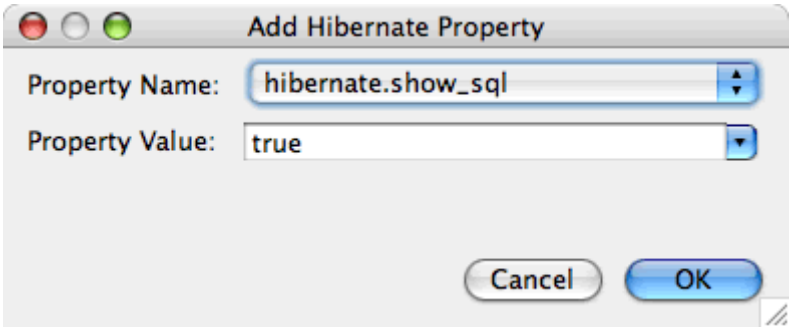
VEURE LES CONSULTES SQL QUE FA HIBERNATE

Per a poder analitzar les consultes que fa Hibernate i veure quines no són correctes podem activar una opció que faci que Hibernate ens mostri per la sortida estàndard totes les consultes que fa en SQL.

Per això obrim el fitxer Hibernate.cfg.XML en el mode de disseny. Hauríem de tenir una finestra com la mostrada a continuació (la configuració del fitxer pot variar).



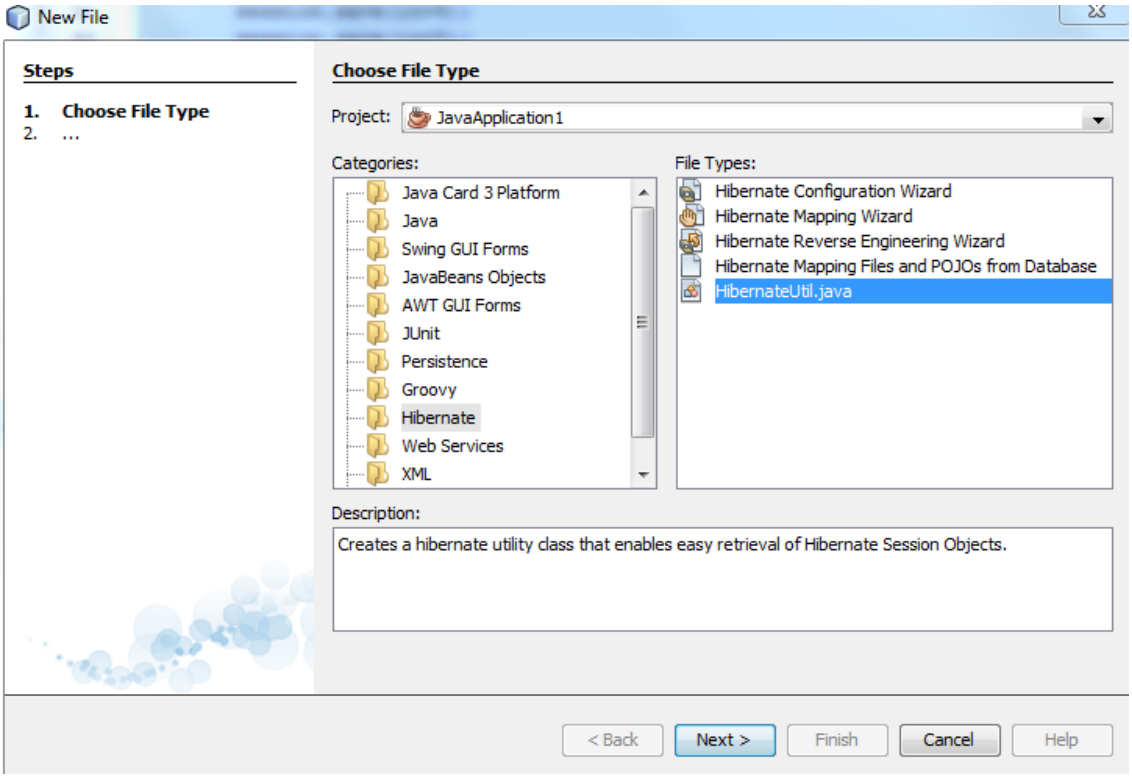
A continuació expandim les propietats de configuració (*'Configuration Properties'*) i tot seguit les opcionals (*'Optional Properties'*). Aquí cliquem afegir i seleccionem *Hibernate.show_sql* tal i com mostra la següent figura. Finalment acceptem els canvis.



CLASSE D'UTILITATS DE HIBERNATE

Aquesta classe ens crea un mètode convenient per a poder iniciar una transacció amb la base de dades anomenat *getSessionFactory*.

Per a crear el fitxer és tant senzill com afegir un nou fitxer al projecte utilitzant la plantilla de *HibernateUtil* tal i com mostra la següent pantalla. A diferència del fitxer de configuració, aquest pot estar situat en qualssevol localització i pot tenir qualssevol nom.

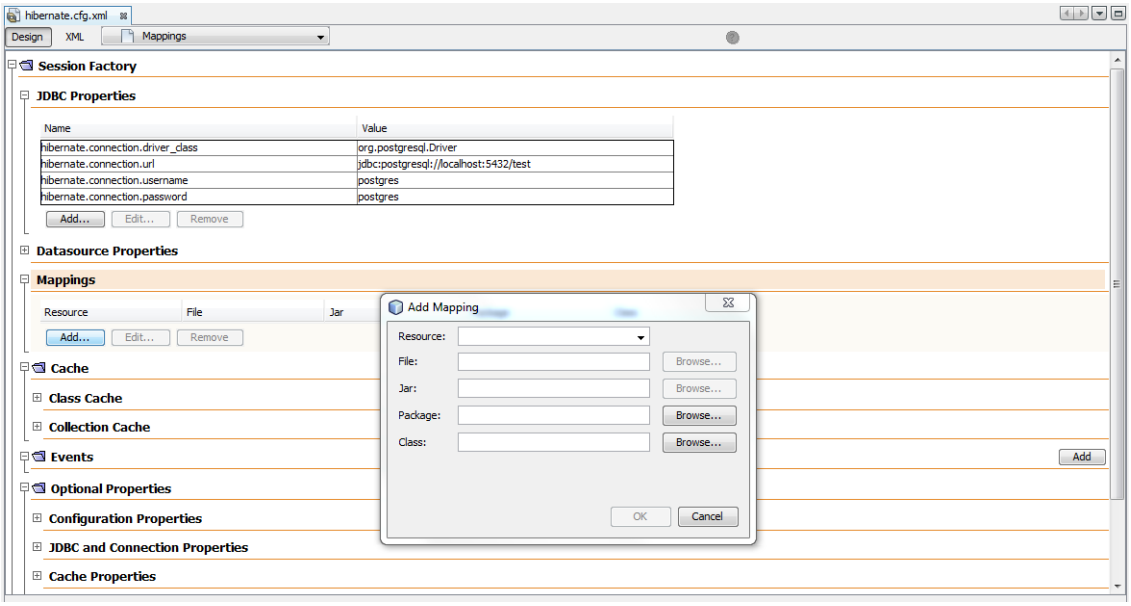


ADDICIÓ D'UNA CLASSE

Per a cada classe que afegim i vulguem que Hibernate la tingui en compte, s'ha de fer el següent:

Obrir el fitxer Hibernate.cfg.XML amb l'editor del NetBeans (en mode disseny). Expandir mapeigs i clicar a *afegir*. Això s'haurà de fer per a cada classe que es vulgui tenir en compte per a la persistència (com més elements es posin, més tardarà en inferir els tipus i, per tant, en "compilar"). Cal tenir en compte que el *refactor* no té en compte això d'aquí, així que si es canvia el nom de la classe s'haurà d'editar el camp corresponent al fitxer de configuració manualment. **Les subclasses també s'han d'afegir.**

Com es pot observar en la següent figura, el paquet és indiferent, ja que Hibernate només tractarà les classes especificades, no pas paquets complets.



FAQ

Tot i que tal i com ja s'ha comentat Hibernate compleixi l'estàndard JPA, aquest també l'amplia. En aquest apartat no tractarem sobre les extensions de Hibernate (que podeu trobar a: http://docs.jboss.org/Hibernate/stable/annotations/reference/en/html_single/#Entity-hibspec) sinó que tractarem sobre alguns paràmetres de la configuració que us poden ser útils:

CONFIGURACIÓ DE L'ESQUEMA

Per defecte Hibernate no crea les taules en l'esquema seleccionat a la configuració d'aquest primer, així que si aquestes no existeixen donarà error. Per a indiciar a Hibernate que creï les taules tenim diverses opcions, totes elles estan en l'opció hbm2ddl.auto (NetBeans: Hibernate.cfg.XML => *Optional Properties* => *Miscellaneous*, Eclipse: Hibernate.cfg.XML => *Properties* => Hibernate => *Hbm2ddl Auto*)

validate: valida l'esquema, no realitza cap canvi a la base de dades.

update: actualitza l'esquema.

create: crea l'esquema eliminant les dades que hi puguin haver.

create-drop: elimina l'esquema al final de la sessió.

L'opció recomanada és *Update*.

NO EM TROBA EL FITXER "HIBERNATE.CFG.XML"

Assegura't que el fitxer està al nivell superior de l'aplicació, això és, al paquet per defecte (*Default Package*).

QUAN CANVIO DE DATA MAPPER NO EM FUNCIONA

Assegura't que no utilitzes cap anotació de *org.hibernate.**. Per a que l'aplicació sigui purament JPA totes les anotacions haurien de provenir de *javax.persistence.**.

JPA

Hi ha dos models d'utilització de Hibernate, seguint l'estàndard JPA, per fitxers XML o anotacions en els fitxers Java. En aquesta assignatura només tractarem el segon, degut a la seva familiaritat amb tots els conceptes apresos fins ara. Cal tenir en compte, però, que hi ha determinades accions que no s'utilitzaran en aquesta assignatura i que és possible que només es puguin realitzar mitjançant la configuració XML.

JPA està definida per a JEE i JSE, aquí només tractarem la segona. Això és, que utilitzarem només *POJO's* (*Plain Old Java Objects*) i no utilitzarem els *Java Beans* de *JEE*. Aquest és un factor d'extrema importància perquè hi ha elements de JPA que no estan disponibles per a JSE, per exemple *@MapsId*. Tot i que aquests elements estan sovint implementats pels propis Data Mappers, per a evitar la limitació del JPA, cal recordar que les implementacions varien i que, per tant, alguns pedaços que es puguin aplicar a Hibernate no estaran disponibles a d'altres Data Mappers o tindran un ús diferent.

INTRODUCCIÓ

Hibernate és un Data Mapper com molts d'altres. Aquest és un breu resum de l'ús de Hibernate mitjançant anotacions. Cal dir que com que estem treballant en Java, tot i mencionar pràcticament sempre Hibernate, la implementació de les anotacions i dels XML és seguint l'estàndard JPA. Hibernate disposa d'unes extensions al estàndard que no seran tractades aquí, per a més informació es pot cercar als apèndixs.

ENTITAT (ENTITY)

Cada Entitat és representada com una taula en la base de dades, per tant només les classes poden ser del tipus Entitat.

@Entity és l'identificador que s'ha d'afegir a una classe per a que Hibernate la processi, altrament Hibernate no tractarà aquella classe (és a dir, no llegirà cap anotació).

Per a modificar els atributs d'una Entitat utilitzem *@Table*, que ens permet modificar diversos elements (veure: http://docs.oracle.com/Javaee/5/api/Javax/persistence/Table.html).

Cada Entitat per definició ha de tenir una clau primària, que pot ser simple o composta (les segones les tractarem més endavant). Per a definir una clau primària simple és tant senzill com afegir l'anotació *@Id* davant de la declaració de l'element. Qualssevol element pot ser una clau primària, incloent les relacions.

Una Entitat simple podria ser:

```
@Entity

@Table(name="TRANSPORTS")

Class transports {

        @Id int transport_id;

}
```

Cal recordar que per a que Hibernate tracti una classe fa falta afegir el mapeig a Hibernate.cfg.XML apart d'afegir l'anotació *@Entity*.

HERÈNCIA

Hi ha tres tipus d'herència en JPA, a continuació es farà un breu repàs d'aquests:

SINGLE_COLUMN: La classe i les seves subclasses comparteixen una mateixa taula, això vol dir que la taula tindrà tantes columnes com la suma de les de la classe i les de les seves subclasses, essent nul·les o tenint un valor diferenciant en cas que no s'utilitzin.

JOINED: Cada classe té la seva pròpia taula on els propis elements de la classe i els de les seves subclasses hi tenen una entrada. Cada subclasse, a més, té la seva pròpia taula on hi haurà una columna que farà referència a la clau primària de la superclasse, apart hi guardarà els seus atributs persistents.

TABLE_PER_CLASS: Cada classe té una taula per a ella sola, independentment de si té o no té subclasses. Cada subclasse tindrà les seves columnes i les de la superclasse.

Només s'explicarà amb detall el tipus JOINED ja que els altres són com ja s'ha explicat a classe.

<Esquema d'herència: P1 i P2 hereten de P, les taules tindran els següents atributs: p => string nom, string extra; p1 => int diners; p2 => <res>. Per a cada taula hi ha un camp que és l'identificador únic en la base de dades, de tipus enter; per a p1 i p2 hi un camp extra a la seva taula que és l'identificador de p.>

Com es pot observar en l'esquema anterior cada element de p1 i p2 tindran una entrada a p a més de la seva corresponent a p1 o p2. Per a cercar si un element donat de p és realment p1 o p2 s'haurà de cercar en ambdues taules per a trobar si hi ha alguna entrada que tingui el mateix *identificador_p* de l'entrada cercada a p. Això és altament ineficient així que JPA ens ofereix la possibilitat d'utilitzar columnes discriminants, la declaració és citada a continuació.

@DiscriminatorColumn(name="TIPUS_P", discriminatorType=DiscriminatorType.STRING, length=50)

La següent anotació s'ha de posar a cada subclasse si es fa servir les columnes discriminants.

@DiscriminatorValue("P")

Com es pot suposar les columnes discriminants equivalen a afegir una columna a la taula de p que indiqui el valor discriminant (per exemple, es podrien definir com els tipus P per a p, P1 per a p1 i P2 per a p2). Els valors permesos són *string*, *char* i *int*. D'aquesta forma en cada classe que hereti de p (p inclosa) s'haurà de posar un valor discriminant, i diferent entre ells, que serà el que hi constarà a l'entrada quan es creï un element de p o de les seves subclasses.

CLAUS

CLAU FORANA (FK)

Qualssevol relació implica una clau forana però amb JPA aquestes queden amagades sota la sintaxis utilitzada per a definir una relació. <res mes!>

CLAUS COMPOSTES(ENUMERACIONS)

Abans ja hem explicat que les claus poden ser simples o compostes i el funcionament de les primeres. Les claus compostes tenen dues formes d'implementar-se, a continuació s'explicaran ambdós.

DEFINIDES EN LA PRÒPIA CLASSE

D'aquesta forma per a cada element que hagi de formar part de la clau primària se li haurà d'afegir un *@Id* al davant de la seva declaració. Amb això tindrem la clau primària definida amb elements compostos, però no podrem separar dos elements entre ells ja que al haver-hi diverses claus primàries Hibernate no sabrà com a diferenciar-lo. Per a solucionar aquest aspecte, cal notar que això s'ha de fer obligatòriament si s'empra aquest mètode, s'haurà d'implementar una classe auxiliar que tingui com a mínim els mateixos camps que són clau primària de l'element original i implementar-ne els mètodes *equals* i *hashcode* de la classe auxiliar (utilitzant exclusivament, és clar, els elements de la clau primària per a obtenir ambdós). Finalment s'haurà d'afegir l'anotació *@IdClass(<nom_classe_auxiliar>.class*) a la classe original. Cal dir que la classe auxiliar pot disposar de més mètodes si s'escau.

DEFINIDES EN UNA CLASSE AUXILIAR

Això és, crear una classe auxiliar i definir-ne allà els camps que s'utilitzaran de clau primària, i només aquests. Es poden definir altres mètodes si s'escau, però no fa falta implementar els mètodes *equals* i *hashcode*.

A la classe principal s'haurà de posar com a clau primària una instància del tipus auxiliar (per exemple, *@Id private TransportPK transport_id*), mentre que a la classe auxiliar se l'hi haurà d'afegir el camp *@Embeddable*.

DIFERÈNCIES ENTRE AMBDUES

La disposició estructural és exactament la mateixa, només canvia l'estructura al fer consultes. Per exemple:

Solució 1:

select t.id from Transport T

En la solució 1, *TransportPK* actua com a comparador per a la mateixa classe, els únics mètodes que importen aleshores són el mètode *equals* i el mètode *hashcode*.

Solució 2:

select t.transport_id.id from Transport t

En la solució 2, *TransportPK* és realment una enumeració i actua com a tal.

RELACIONS

Totes les relacions són unidireccionals, si es vol que siguin bidireccionals s'hauran de declarar també en la classe destí.

@MANYTOMANY (*-*)

Aquí s'explicaran les relacions genèriqies * - * i també les relacions *n..m - **.

El primer cas és realment senzill, com anirem veient amb les altres relacions, només fa falta declarar una col·lecció d'elements per a que la relació funcioni, així:

@ManyToMany private Collection<Transport> transports

En les relacions hi podem definir paràmetres, per exemple *@ManyToMany(cascade={CascadeType.ALL})* indicaria que si es produeix una actualització aquesta afectarà a tots els elements que hi estiguin relacionats. Per a veure tots els elements consultar al *JavaDocs* de JPA.

@ONETOONE (1-1)

Aquest tipus de relació no requereix de col·lecció, per tant la declaració pot ser de la forma *@OneToOne private Transport transport*. Per a canviar atributs podem utilitzar *@JoinColumn*.

@MANYTOONE (*-1)

Igual que la relació anterior, s'empraria seria *@ManyToOne private Transport transport*.

@ONETOMANY (1-*)

Aquest seguiria l'exemple anterior de *ManyToMany* però utilitzant una col·lecció, a continuació se'n mostra un exemple. *@ManyToMany private Collection<Transport> transports*.

RELACIONS M..N

En SQL no hi ha forma de tractar aquest tipus de relacions, de forma que aquestes hauran de ser tractades en la lògica del programa. Per a tal es pot fer, per exemple, una classe que hereti de *Collection* i que tracti els elements. Disposeu d'un exemple en la classe *SampleNHashSet*.

CONSULTES

ACTUALITZACIONS A LA BASE DE DADES

Per a realitzar consultes es recomana crear una classe d'ajuda (podeu utilitzar la plantilla *HibernateUtil* en NetBeans, a Eclipse ho haureu de fer manualment). En els exemples utilitzats a continuació s'assumeix que s'ha utilitzat la plantilla i, per tant, es disposa del mètode *getSessionFactory*.

En JPA les consultes es realitzen amb el patró transacció, això és, s'obre una sessió a la base de dades i es guarden els elements que es desitgin. Un cop finalitzat es fa *Commit* de la transacció i es tanca la sessió. A continuació podem veure un exemple:

```
SessionFactory factory = Util.getSessionFactory();

Session session = factory.openSession();

session.beginTransaction();

Transport t = new Transport("Bicicleta");

session.save(t);

session.getTransaction().commit();
```

Si es produeix algun error SQL aquest ens serà llençat al fer el *Commit*. Tenir en compte que l'error no serà del tipus *SQLException* ja que les transaccions es realitzen en format cua de treball, així que s'haurà de tractar adequadament (depenent del Data Mapper serà del tipus *BatchSQLException*, que contindrà com a excepcions internes totes les excepcions SQL que s'hagin pogut llançar).

CONSULTES

Hi ha dos tipus de consultes, tant utilitzant SQL com HQL, nosaltres només tractarem com utilitzar les consultes en SQL. Aquí s'explicarà com obtenir resultats en diferents formats.

Per a executar una consulta SQL només cal utilitzar el mètode *session.createSQLQuery(<consulta>)* on *<consulta>* és la sentència SQL. A continuació es llista les formes d'obtenir resultats en dos formats molt utilitzats.

- Llista (Hibernate inferirà els tipus retornats, però també es poden dir amb *.list().afegirScalar(Tipus 1).afegirScalar(Tipus2)...*)
  - *Session.createSQLQuery("SELECT * FROM TRANSPORTS").list();*
  - Retorna *Object[]*.
- Entity
  - *Session.createSQLQuery("SELECT * FROM TRANSPORTS").afegirEntity(Transport.class);*
  - Retorna l'Entitat, en aquest cas un objecte del tipus *Transport.*
- Una llista sencera dels formats possibles es pot trobar a http://docs.jboss.org/Hibernate/orm/3.3/reference/en/html/querysql.html