

***Ndour Coumba
Totaro Pierre-A.***

***Encadrants: Olivier Barais - Manuel Leduc
Année 2015/2016***

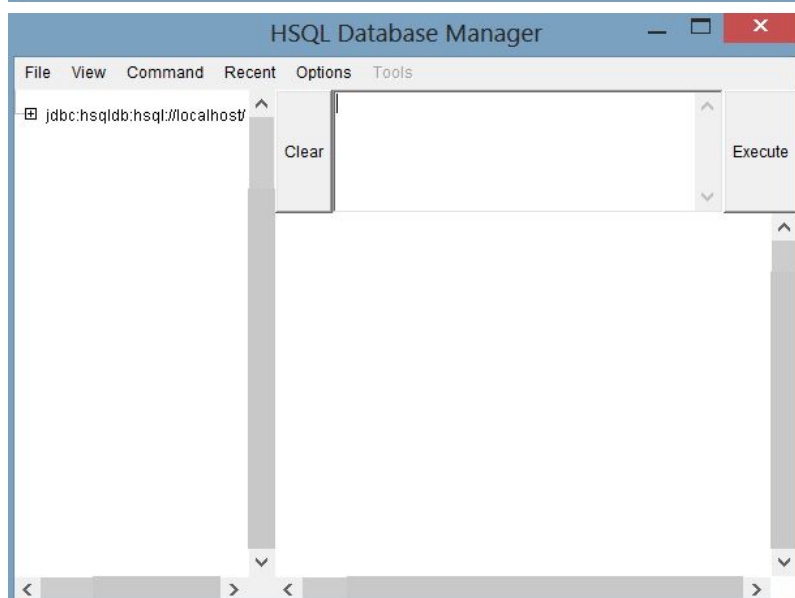
M1 MIAGE

TP JPA et SERVLET - SIR

Pour démarrer l'interaction entre l'application java et la base de données, nous utiliserons le Manager, fourni dans le TP.

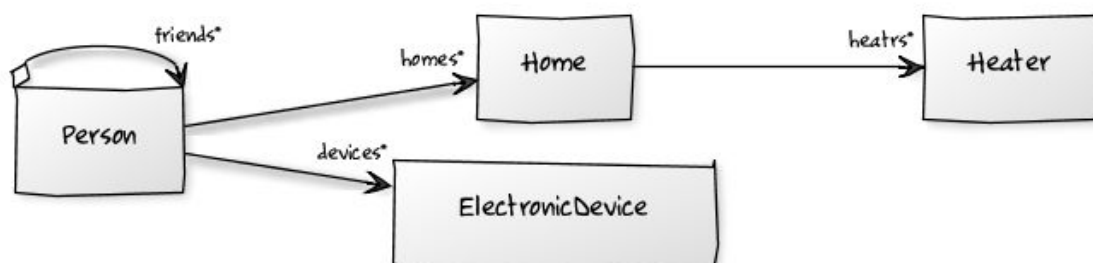
En lançant les deux scripts, on obtient ces deux fenêtres:

```
C:\Users\pitou35\git\sir-tp3suite\data>java -cp ..\dependency\hsqldb-2.3.3.jar org.hsqldb.Server
[Server@28a418fc]: [Thread[main,5,main]]: checkRunning(false) entered
[Server@28a418fc]: [Thread[main,5,main]]: checkRunning(false) exited
[Server@28a418fc]: Startup sequence initiated from main() method
[Server@28a418fc]: Could not load properties from file
[Server@28a418fc]: Using cli/default properties only
[Server@28a418fc]: Initiating startup sequence...
[Server@28a418fc]: Server socket opened successfully in 5 ms.
[Server@28a418fc]: Database [index=0, id=0, db=file:test, alias=] opened successfully in 510 ms.
[Server@28a418fc]: Startup sequence completed in 516 ms.
[Server@28a418fc]: 2016-02-19 14:40:22.399 HSQLDB server 2.3.3 is online on port 9001
[Server@28a418fc]: To close normally, connect and execute SHUTDOWN SQL
[Server@28a418fc]: From command line, use [Ctrl]+[C] to abort abruptly
```



C'est dans cette dernière que nous pourrions visualiser notre base de données.

L'objectif serait de créer des entités respectant ce schéma:



Nous créons donc les entités Person, Home, Heater, ElectronicDevice dans des classes java distinctes.

Puis, dans la classe JPATest, nous pouvons donc instancier des objets, et les envoyer dans le manager.

```

try {
    Person p1 = new Person("pitou", "toutou", "@france.fr");
    Person p2 = new Person("marco", "polo", "@italy.it");
    Home h1=new Home(20,20);
    h1.setOwner(p1);
    ElectronicDevices e1=new ElectronicDevices(20);
    ElectronicDevices e2=new ElectronicDevices(20);
    ElectronicDevices e3=new ElectronicDevices(20);
    ElectronicDevices e4=new ElectronicDevices(20);
    e1.setOwner(p1);
    e2.setOwner(p1);
    e3.setOwner(p1);
    e4.setOwner(p1);
    Heater he1=new Heater(1800);
    Heater he2=new Heater(1800);
    Heater he3=new Heater(1800);
    Heater he4=new Heater(1800);
    he1.setOwner(h1);
    he2.setOwner(h1);
    he3.setOwner(h1);
    he4.setOwner(h1);

    manager.persist(p1);
    manager.persist(p2);
    manager.persist(h1);
    manager.persist(he1);
    manager.persist(he2);
    manager.persist(he3);
    manager.persist(he4);
    manager.persist(e1);
    manager.persist(e2);
    manager.persist(e3);
    manager.persist(e4);
}

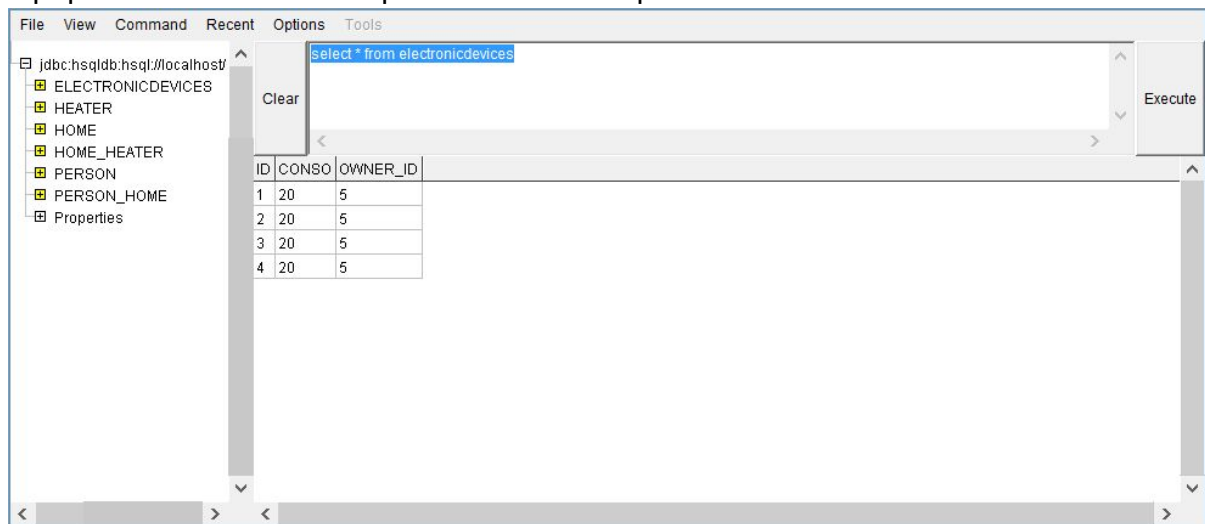
```

Dans le cas présent, nous créons deux personnes. Nous nous occuperons exclusivement de la première s'appelant 'pitou'. Elle possède une maison et quatre gadgets mobiles. Sa maison possède quatre chauffages.

Si l'on exécute ce JpaTest.java, notre manager va se remplir. Nous pouvons le vérifier avec quelques requêtes SQL.



Ici, nos deux personnes sont bien créées. Nous pourrions aussi lister les équipements mobiles de 'pitou' dont la clé primaire est '5'.



Nous visualisons bien nos quatre équipements.

Connexion à une base MySQL

Maintenant, nous allons abandonner le manager et utiliser une base MySQL. Nous adopterons une base en local, il faut donc renseigner certaines informations dans le fichier persistence.xml:

```
<persistence-unit name="mysql">
  <properties>
    <!--
      <property name="hibernate.ejb.cfgfile" value="/hibernate.cfg.xml"/>
      <property name="hibernate.hbm2ddl.auto" value="create"/>
    -->
    <property name="hibernate.hbm2ddl.auto" value="create"/>
    <property name="hibernate.archive.autodetection" value="class, hbm"/>
    <property name="hibernate.show_sql" value="true"/>
    <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
    <property name="hibernate.connection.password" value=""/>
    <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/sir"/>
    <property name="hibernate.connection.username" value="root"/>
    <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
    <property name="hibernate.c3p0.min_size" value="5"/>
    <property name="hibernate.c3p0.max_size" value="20"/>
    <property name="hibernate.c3p0.timeout" value="300"/>
    <property name="hibernate.c3p0.max_statements" value="50"/>
    <property name="hibernate.c3p0.idle_test_period" value="3000"/>
  </properties>
</persistence-unit>
```

D'après les paramètres, nous sommes donc en localhost sur la base "sir" avec l'identifiant "root" sans mot de passe. Nous sommes en option "create" qui supprimera à chaque fois la base de données avant de la remplir à nouveau, dès que l'on lancera le JPATest.java

Il ne faut pas oublier d'importer le driver jdbc vers mysql en l'ajoutant dans pom.xml, section lue par l'outil Maven.

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.36</version>
</dependency>
```

Avant d'exécuter, on n'oublie pas de changer la source du manager qui ne sera plus "dev" mais "mysql", afin d'aller lire la bonne section du persistence.xml

```
public static void main(String[] args) {
    EntityManagerFactory factory = Persistence
        .createEntityManagerFactory("mysql");
    EntityManager manager = factory.createEntityManager();
    JpaTest test = new JpaTest(manager);
    EntityTransaction tx = manager.getTransaction();
    tx.begin();
    try {

        Person p1 = new Person("pitou", "toutou", "@france.fr");
        Person p2 = new Person("marco", "polo", "@italy.it");
        Home h1=new Home(20,20);
```

Une fois lancé, nous pourrions constater que la BDD mySQL sera remplie de la même manière que le manager JDBC l'a été. Au préalable, nous pourrions visualiser les logs d'Hibernate.

```
log4j:WARN No appenders could be found for logger (org.jboss.logging).
log4j:WARN Please initialize the log4j system properly.
Hibernate: alter table ElectronicDevices drop foreign key FK_4y9k3leviekbixoxslkaxbuch
Hibernate: alter table Heater drop foreign key FK_dqkh5wieaagg7f2qu8bkxoq4a
Hibernate: alter table Home drop foreign key FK_bbn13mgx1n7k7uj7gt0eh9tdd
Hibernate: drop table if exists ElectronicDevices
Hibernate: drop table if exists Heater
Hibernate: drop table if exists Home
Hibernate: drop table if exists Person
Hibernate: create table ElectronicDevices (id integer not null auto_increment, conso integer not null, OWNER_ID integer, primary key (id))
Hibernate: create table Heater (id integer not null auto_increment, conso integer not null, OWNER_ID integer, primary key (id))
Hibernate: create table Home (id integer not null auto_increment, nbPieces integer not null, taille integer not null, OWNER_ID integer, primary key (id))
Hibernate: create table Person (id integer not null auto_increment, email varchar(255), nom varchar(255), prenom varchar(255), primary key (id))
Hibernate: alter table ElectronicDevices add constraint FK_4y9k3leviekbixoxslkaxbuch foreign key (OWNER_ID) references Person (id)
Hibernate: alter table Heater add constraint FK_dqkh5wieaagg7f2qu8bkxoq4a foreign key (OWNER_ID) references Home (id)
Hibernate: alter table Home add constraint FK_bbn13mgx1n7k7uj7gt0eh9tdd foreign key (OWNER_ID) references Person (id)
Hibernate: insert into Person (email, nom, prenom) values (?, ?, ?)
Hibernate: insert into Person (email, nom, prenom) values (?, ?, ?)
Hibernate: insert into Home (nbPieces, OWNER_ID, taille) values (?, ?, ?)
Hibernate: insert into Heater (conso, OWNER_ID) values (?, ?)
Hibernate: insert into Heater (conso, OWNER_ID) values (?, ?)
Hibernate: insert into Heater (conso, OWNER_ID) values (?, ?)
Hibernate: insert into Heater (conso, OWNER_ID) values (?, ?)
Hibernate: insert into ElectronicDevices (conso, OWNER_ID) values (?, ?)
Hibernate: insert into ElectronicDevices (conso, OWNER_ID) values (?, ?)
Hibernate: insert into ElectronicDevices (conso, OWNER_ID) values (?, ?)
Hibernate: insert into ElectronicDevices (conso, OWNER_ID) values (?, ?)
```

Table	Action	Lignes	Type	Interclassement	Taille	Perte
electronicdevices	Afficher Structure Rechercher Insérer Vider Supprimer	~4	InnoDB	latin1_swedish_ci	32 Kio	-
heater	Afficher Structure Rechercher Insérer Vider Supprimer	~4	InnoDB	latin1_swedish_ci	32 Kio	-
home	Afficher Structure Rechercher Insérer Vider Supprimer	~1	InnoDB	latin1_swedish_ci	32 Kio	-
person	Afficher Structure Rechercher Insérer Vider Supprimer	~2	InnoDB	latin1_swedish_ci	16 Kio	-
4 tables	Somme	11	InnoDB	latin1_swedish_ci	112 Kio	0 o

Ici, visualisation des tables créées.


```
SELECT *
FROM `electronicdevices`
LIMIT 0 , 30
```

Afficher : Ligne de départ: 0 Nombre de lignes: 30 En-têtes à chaque 100 ligne

Trier sur l'index: Aucune

+ Options

				id	conso	OWNER_ID
<input type="checkbox"/>		Modifier		Copier		Effacer
				1	20	1
<input type="checkbox"/>		Modifier		Copier		Effacer
				2	20	1
<input type="checkbox"/>		Modifier		Copier		Effacer
				3	20	1
<input type="checkbox"/>		Modifier		Copier		Effacer
				4	20	1

Et là, visualisation du contenu de la table ElectronicDevices.

Utilisation de Servlet

Maintenant//Maintenant que l'application JPA fonctionne, il faut ajouter des servlet.

Pour cela, nous devons ajouter une dépendance dans le pom.xml

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.0.1</version>
  <scope>provided</scope>
</dependency>
```

Nous pouvons lancer "mvn tomcat7:run" pour vérifier si notre configuration est correcte. D'après la console ci-dessous, cela semble être le cas.

```
Problems @ Javadoc Declaration Console Progress Properties
sir-tp3suite [Maven Build] C:\Program Files\Java\jdk1.7.0_79\jre\bin\javaw.exe (19 févr. 2016 21:15:29)
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ testjpa ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent!
[INFO] Compiling 9 source files to C:\Users\pitou35\git\sir-tp3suite\target\classes
[INFO]
[INFO] <<< tomcat7-maven-plugin:2.2:run (default-cli) < process-classes @ testjpa <<<
[INFO]
[INFO] --- tomcat7-maven-plugin:2.2:run (default-cli) @ testjpa ---
[INFO] Démarrage du war sur http://localhost:8080/
[INFO] Utilisation de la configuration existante du serveur Tomcat sur C:\Users\pitou35\git\sir-tp3suite\target\to
[INFO] create webapp with contextPath:
févr. 19, 2016 9:15:36 PM org.apache.coyote.AbstractProtocol init
INFOS: Initializing ProtocolHandler ["http-bio-8080"]
févr. 19, 2016 9:15:36 PM org.apache.catalina.core.StandardService startInternal
INFOS: Starting service Tomcat
févr. 19, 2016 9:15:36 PM org.apache.catalina.core.StandardEngine startInternal
INFOS: Starting Servlet Engine: Apache Tomcat/7.0.47
févr. 19, 2016 9:15:38 PM com.sun.jersey.api.core.PackagesResourceConfig init
```

Insertion de ressources statiques et consommation d'un formulaire

Maintenant//Il faut maintenant créer un formulaire html qui récupèrera des données saisies dans ce formulaire (ici, la création d'une nouvelle personne), et, après insertion, une page html sera renvoyée contenant les informations transmises auparavant.

Voici les codes correspondants:

Email :

Nom :

Prenom :

```
persistence.xml  TP3Question...  C:\Users\pi...  JpaTest.java

<html>
<body>
<FORM Method="POST" Action="/addPerson">

Email :      <INPUT type=text size=100 name=email><BR>
Nom :      <INPUT type=text size=100 name=nom><BR>
Prenom :    <INPUT type=text size=100 name=prenom><BR>
            <INPUT type=submit value=Send>
</FORM>

@WebServlet(name="addHouse",urlPatterns={"/addPerson"})
public class TP3Question5 extends HttpServlet{

    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        System.out.println("entree");
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        EntityManagerFactory factory2 = Persistence
            .createEntityManagerFactory("mysql");
        EntityManager manager = factory2.createEntityManager();
        EntityTransaction tx = manager.getTransaction();
        tx.begin();

        try {
            Person p1=new Person(request.getParameter("nom"), request.getParameter("prenom"), request.getParameter("email"));
            manager.persist(p1);
        } catch (Exception e) {
            e.printStackTrace();
        }
        tx.commit();
        manager.close();
        factory2.close();
        out.println("<HTML>\n<BODY>\n" +
            "<H1>Recapitulatif des informations</H1>\n" +
            "<UL>\n" +
            "<LI>Email: "
                + request.getParameter("email") + "\n" +
            "<LI>Nom: "
                + request.getParameter("nom") + "\n" +
            "<LI>Prenom: "
                + request.getParameter("prenom") + "\n" +
            "</UL>\n" +
            "</BODY>\n</HTML>");
    }
}
```

Une fois que les informations sont rentrées, la base se voit rajouter une nouvelle personne.

Le test est effectué via l'adresse <http://localhost:8080/FormulaireQuestion5.html>

← → ↻ localhost:8080/FormulaireQuestion5.html

Email :
Nom :
Prenom :

INFO: Initiating Jersey application, version 'Jersey: 1.18.3 12/01/2014 08:23 AM'
févr. 19, 2016 9:42:41 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
entrée
log4j:WARN No appenders could be found for logger (org.jboss.logging).
log4j:WARN Please initialize the log4j system properly.
Hibernate: insert into Person (email, nom, prenom) values (?, ?, ?)

← → ↻ localhost:8080/addPerson




Recapitulatif des informations

- Email: uncertainemail@istic.fr
- Nom: TOTARO
- Prenom: Pierre-Augustin

```
SELECT *  
FROM person  
LIMIT 0, 30
```

Afficher : Ligne de départ: Nombre de lignes: En-têtes à chaque ligne

+ Options

	id	email	nom	prenom
<input type="checkbox"/>  Modifier  Copier  Effacer	2	uncertainemail@istic.fr	TOTARO	Pierre-Augustin

Architecture Rest - Utilisation de Jersey

Maintenant//Maintenant, nous allons utiliser l'outil Jersey pour nous simplifier la création des routes.

On ajoute d'abord les dépendances.

```
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-servlet</artifactId>
  <version>1.18.3</version>
</dependency>

<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-json</artifactId>
  <version>1.18.3</version>
</dependency>
```

Puis on ajoute le descripteur d'application web qui va configurer les servlets dans web.xml

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>MyWebProject</display-name>
  <servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>
        rest
      </param-value>
    </init-param>
    <init-param>
      <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
      <param-value>true</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Et enfin dans notre SampleWebService.java su package Rest, on ajoute ce code pour tester:

```

package rest;

import javax.ws.rs.GET;

@Path("/hello")
public class SampleWebService {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayHello() {
        return "Hello, how are you?";
    }

    @GET
    @Path("/home")
    @Produces(MediaType.APPLICATION_JSON)
    public Home getHome() {
        Home h = new Home(1,1);

        Heater h1 = new Heater(2000);

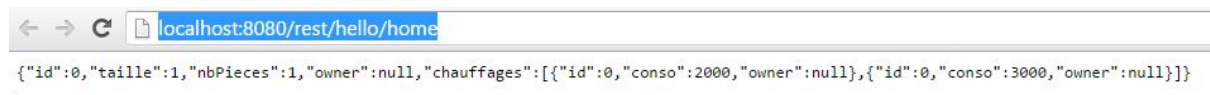
        Heater h2 = new Heater(3000);

        h.addChauffages(h1);
        h.addChauffages(h2);
        return h;
    }
}

```

Une fois dans le navigateur, si l'on saisit la bonne URL:

<http://localhost:8080/rest/hello/home>



Certains attributs sont à "null" car ils représentent des propriétaires qui sont des personnes, entités que nous n'avons pas ici renseignés.