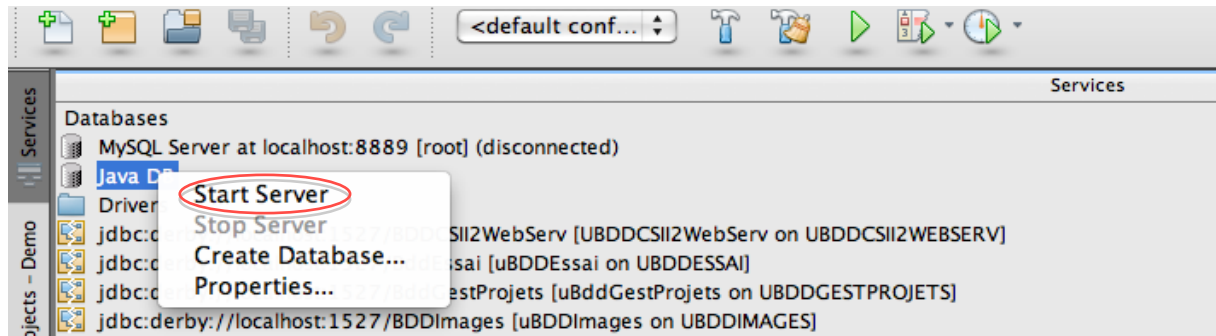


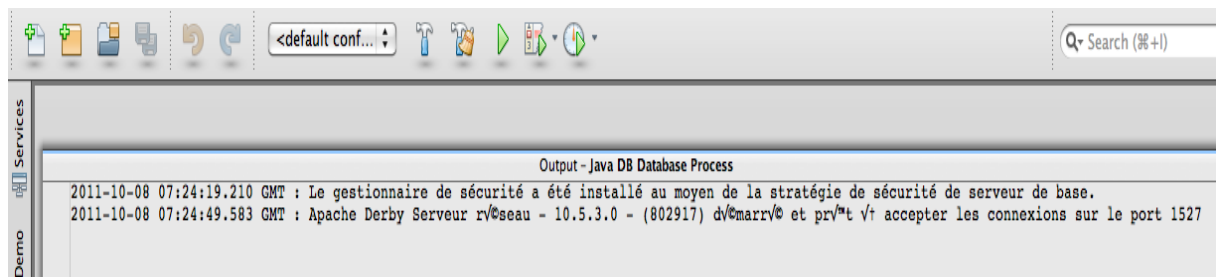
TUTORIEL SUR LA MISE EN OEUVRE DE LA PERSISTANCE JPA AVEC ECLIPSELINK

Pour réaliser ce tutoriel, charger dans Netbeans le projet **TutoJPA**.

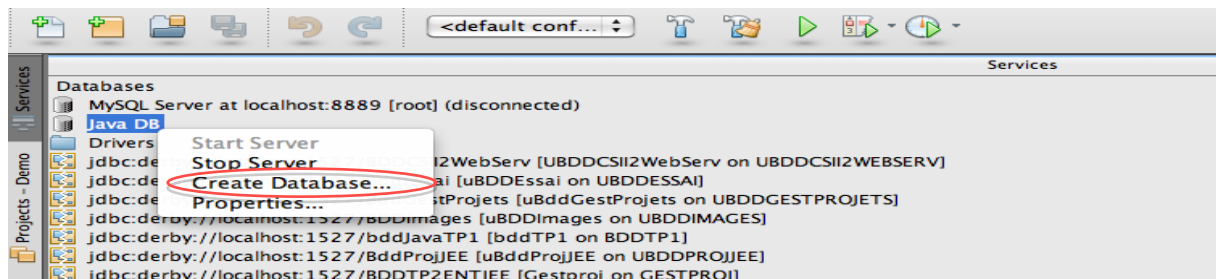
1° Démarrer le serveur de Base de données Java DB (Derby)



Résultat: Vous obtenez un affichage similaire à celui-ci:

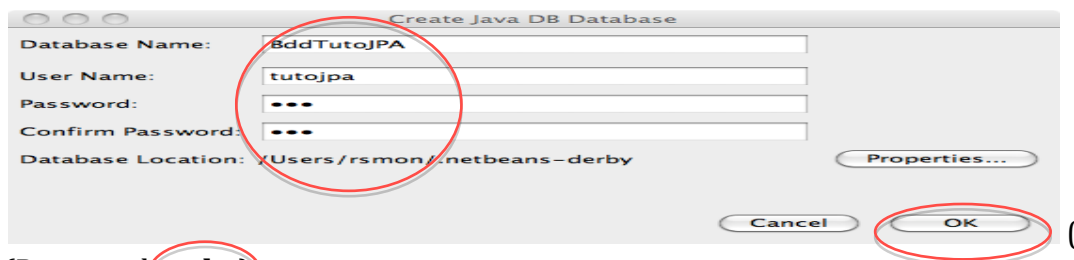


2° Créer une base de données



Sélectionner Create Database...

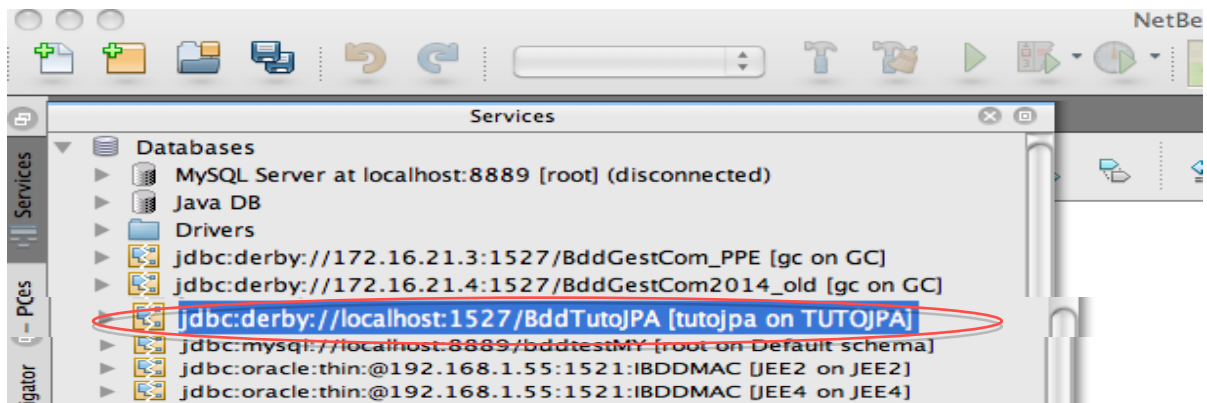
Remplir le formulaire qui apparait avec les informations suivantes:



(Password: **mdp**)

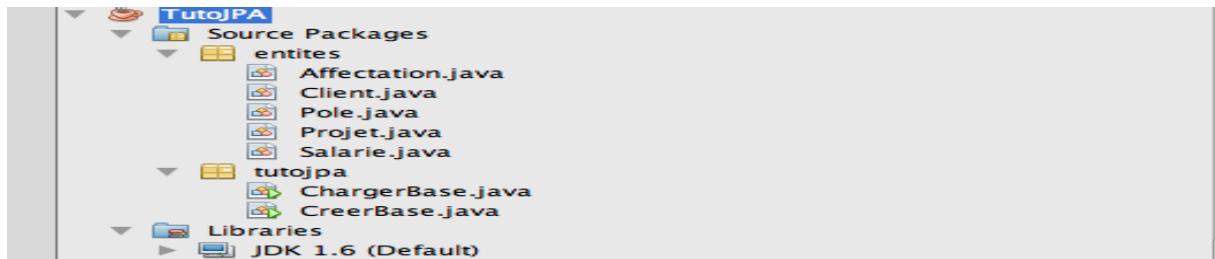
puis accepter avec OK.

On obtient alors une nouvelle base de données comme illustré ci-dessous:



4° Retournons sur le projet **TutoJPA** dans le volet Projets:

Voici son contenu:



Dans ce tutoriel, nous allons travailler avec les classes **Pole** et **Salarie**:

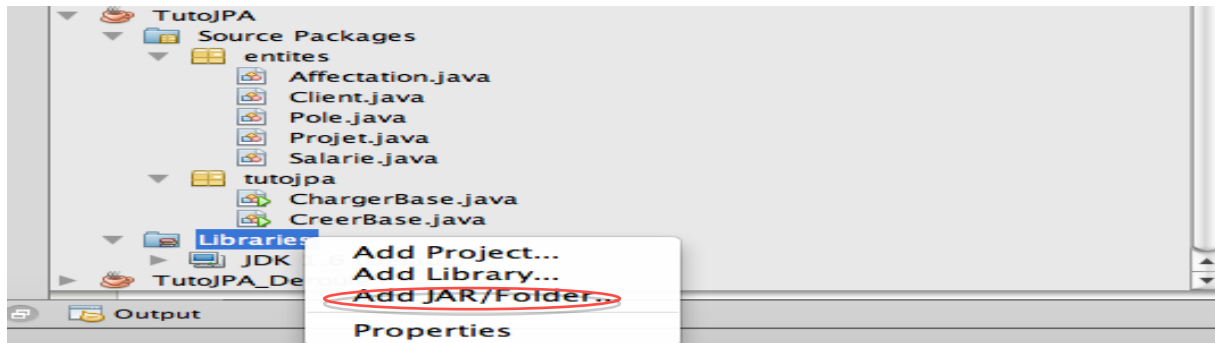
```

1 package entites;
2 public class Pole {
3
4     private String      codePole;
5     private String      nomPole;
6
7     public Pole(String codefonct, String libfonct) {
8
9         this.codePole = codefonct;
10        this.nomPole = libfonct;
11    }
12
13    public void afficher(){
14
15        System.out.print(codePole+ " ");
16        System.out.print(nomPole);
17    }
18
19    Getters et setters
20 }
21
22 package entites;
23 public class Salarie {
24
25     private Long      numsal;
26     private String    nomsal;
27     private String    sexe;
28     private Float     salaire;
29
30     public Salarie( Long numsal, String nomsal, String sexe, Float salaire) {
31
32         this.numsal = numsal;
33         this.nomsal = nomsal;
34         this.sexe = sexe;
35         this.salaire = salaire;
36    }
37
38    public void afficher(){
39
40        System.out.print(numsal+ " ");
41        System.out.print(nomsal+ " ");
42        System.out.print(sexe+ " ");
43        System.out.print(salaire);
44    }
45
46    public void augmenter(Float pctAug){
47
48        salaire=salaire*(1+pctAug/100);
49    }
50
51    Getters et setters
52 }

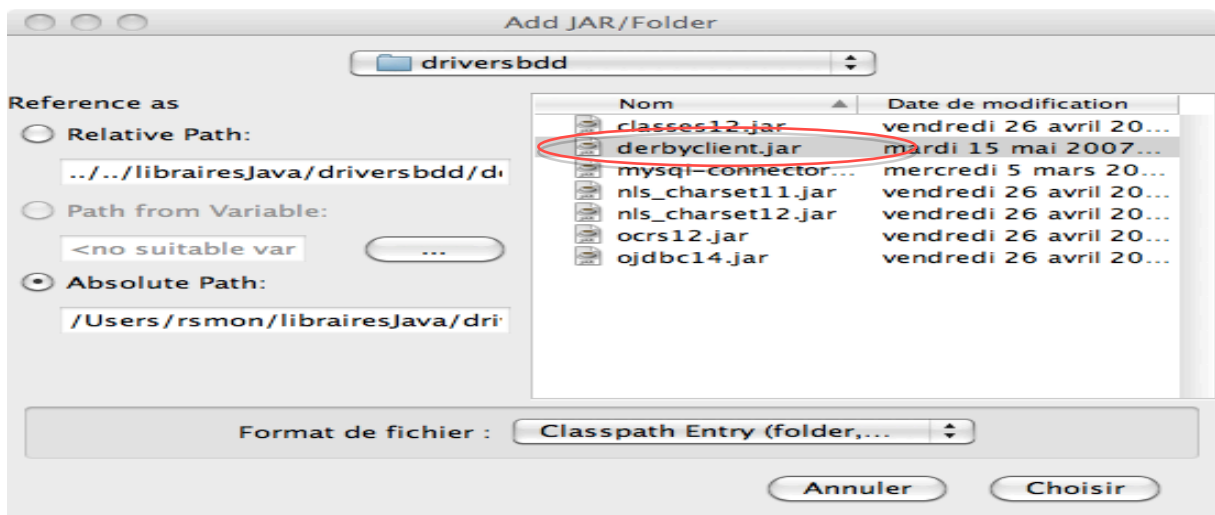
```

5° Notre projet devra utiliser un driver JDBC qui permet d'accéder à la base de données. Il faut donc l'intégrer dans Libraries.

Pour sélectionner **Add JAR/Folder** comme ci-dessous:



6° Sélection du driver **derbyclient.jar** dans le répertoire **driversbdd** fourni

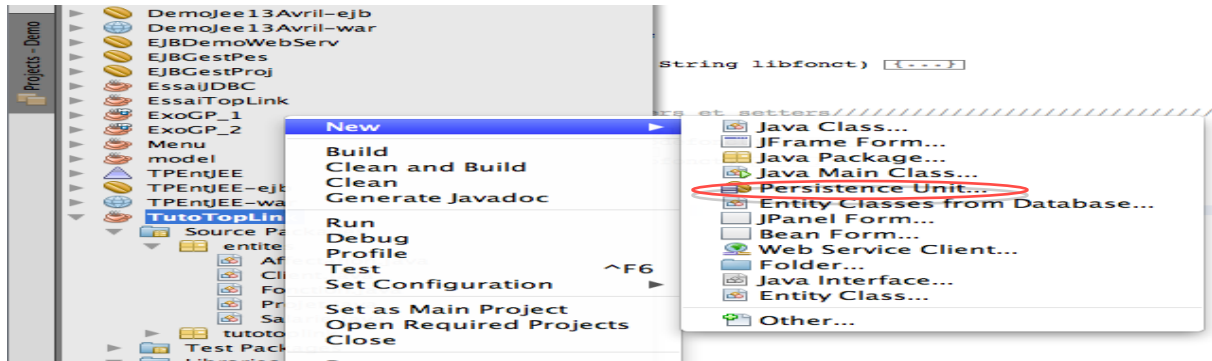


Valider (**Choisir**)

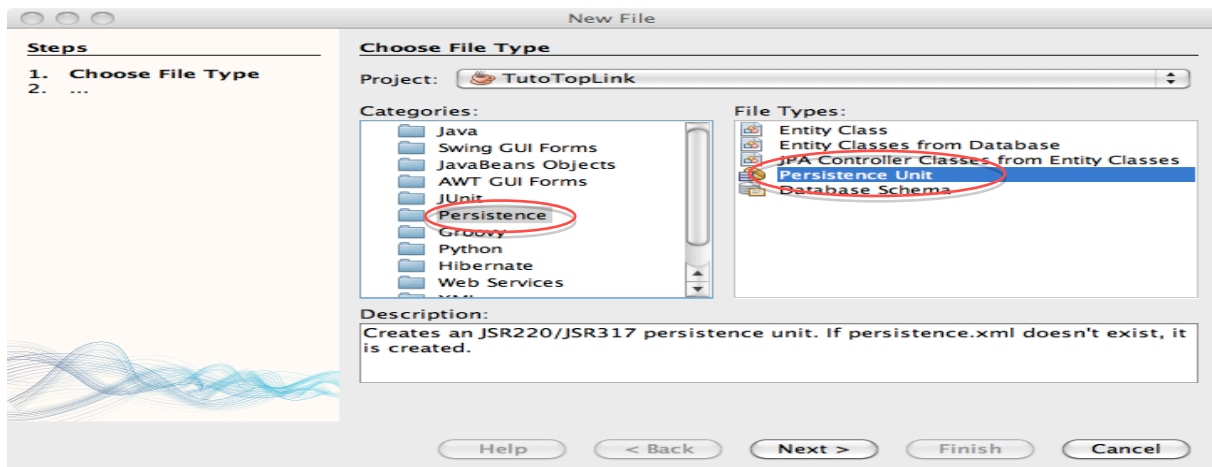
Résultat:



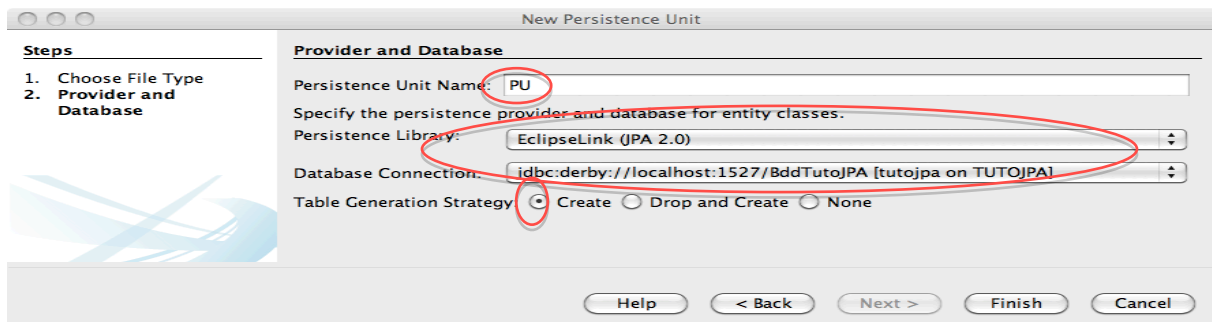
7° Il faut maintenant créer une unité de persistance (Persistence Unit)



(Passer par Others... si Persistence Unit n'apparaît pas dans la liste)



Résultat:



Changer Persistence Unit Name en : **PU**

Changer Persistence Library en : **EclipseLink(JPA 2.0)**

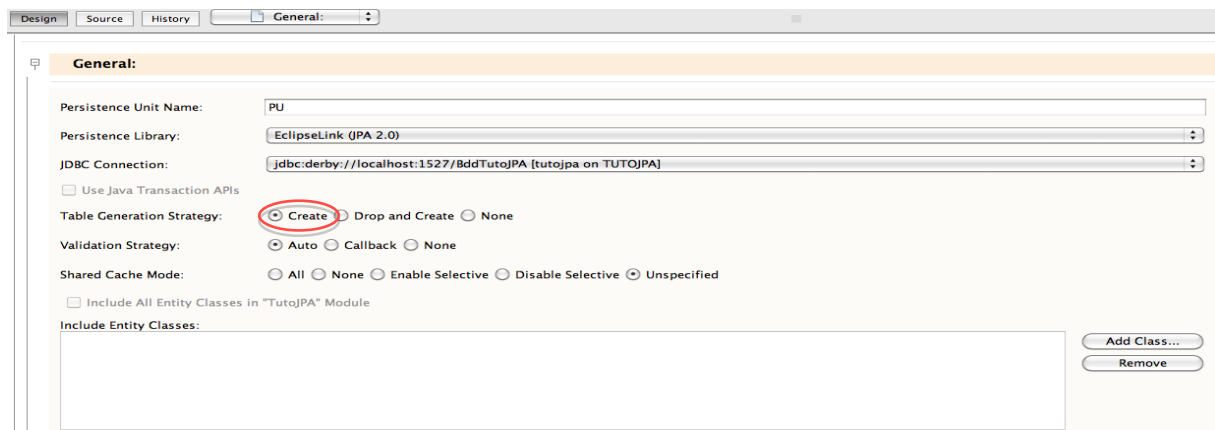
Sélectionner DataBase Connection à :

jdbc:derby://localhost:1527/BddTutoJPA [tutojpa on TUTOJPA]

Laisser Table generation Strategy à : **Create**

Valider avec le bouton **Finish**.

Résultat:

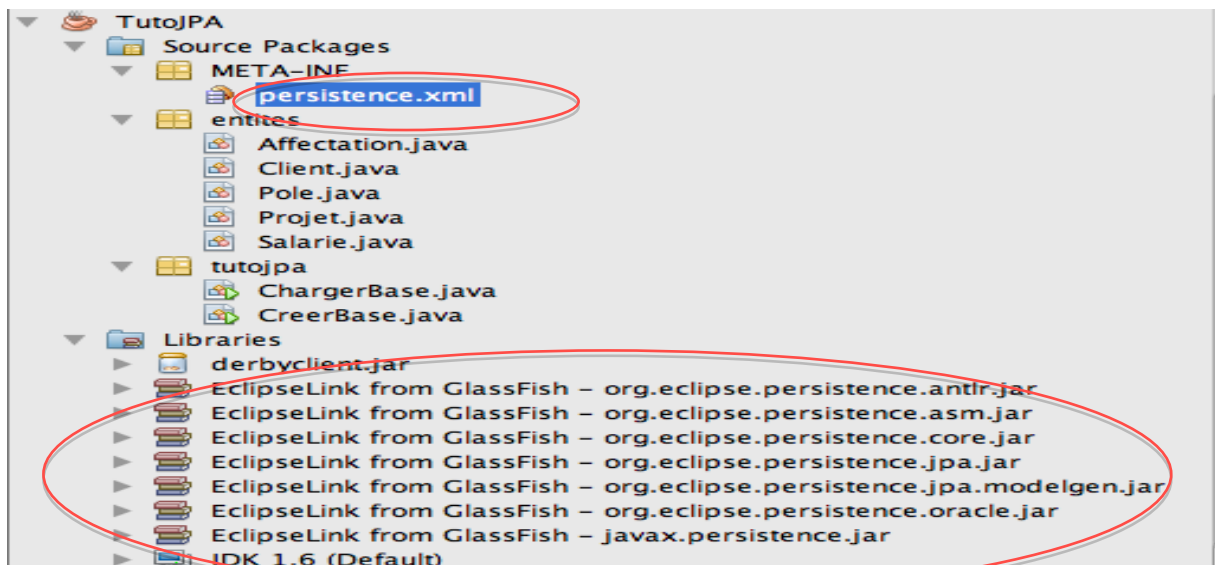


Et si on sélectionne l'onglet **Source** on obtient:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/
3  <persistence-unit name="PU" transaction-type="RESOURCE_LOCAL">
4  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
5  <properties>
6  <property name="javax.persistence.jdbc.url" value="jdbc:derby://localhost:1527/BddTutoJPA"/>
7  <property name="javax.persistence.jdbc.password" value="mdp"/>
8  <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.ClientDriver"/>
9  <property name="javax.persistence.jdbc.user" value="tutojpa"/>
10 <property name="eclipseLink.ddl-generation" value="create-tables"/>
11 </properties>
12 </persistence-unit>
13 </persistence>
    
```

Dans le projet on a maintenant:



Nous allons maintenant ajouter des **annotations JPA** à nos classes **Pole** et **Salarie** et les compléter avec les **attributs navigationnels** prévus dans le diagramme de classes.

Tout au long de ces manipulations l'IDE nous guidera par des suggestions.

8° Faire un **Save(Ctrl S)** et un **Clean and Build** du projet avant de poursuivre.

9° Ajouter l'annotation **@Entity** devant la déclaration de la classe Fonction:

```
1 package entites;
2
3 @Entity
4 public class Pole {
5
6     private String      codePole;
7     private String      nomPole;
8 }
```

Accepter l'import proposé:

```
1 package entites;
2
3 @Entity
4 public class Pole {
5
6     private String      codePole;
7     private String      nomPole;
8
9     public Pole(String codefonct, String libfonct) {
10
11         this.codePole = codefonct;
12         this.nomPole = libfonct;
13     }
14 }
```

...

Résultat:

```
1 package entites;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class Pole {
7
8     private String      codePole;
9     private String      nomPole;
10
11     public Pole(String codefonct, String libfonct) {
12
13         this.codePole = codefonct;
14         this.nomPole = libfonct;
15     }
16
17     public void afficher(){
18
19         System.out.print(codePole+ " ");
20         System.out.print(nomPole);
21     }
22
23     Getters et setters
24 }
25 }
```

Faire la même chose pour la classe **Salarie** pour obtenir:

```
1 package entites;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class Salarie {
7
8     private Long        numsal;
9     private String      nomsal;
10    private String      sexe;
11    private Float        salaire;
12 }
```

....

10° Refaire un **Clean and Build**:

Résultat en observant la classe Pole:

```

1 package entites;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class Pole {
7
8     private String codePole;
9     private String nomPole;
10
11 ...

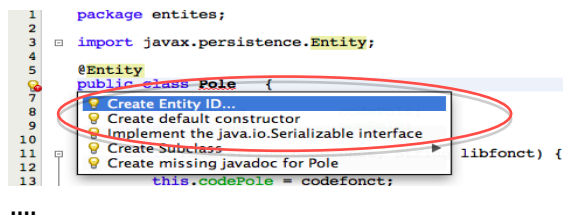
```

Voici ce que suggère l'IDE:

```

1 package entites;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class Pole {
7
8     private String codePole;
9     private String nomPole;
10
11     public Pole(String libfonct) {
12         this.codePole = codefonct;
13     }
14
15 ...

```



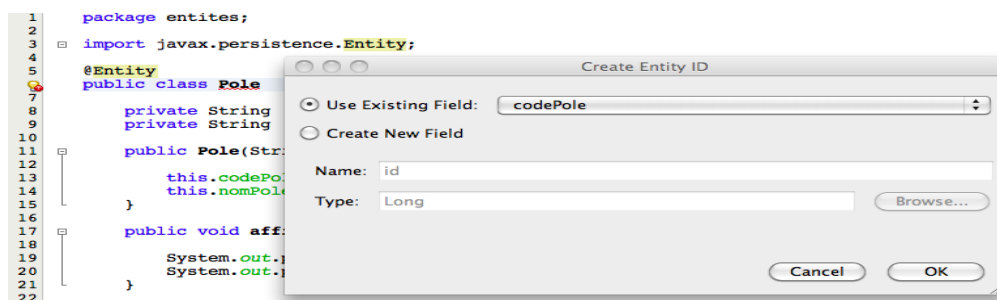
On acceptera les 3 premières suggestions:

D'abord Create Entity ID...

```

1 package entites;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class Pole {
7
8     private String codePole;
9     private String nomPole;
10
11     public Pole(String libfonct) {
12         this.codePole = codefonct;
13     }
14
15 ...

```



Faire OK sans rien changer.

On obtient alors: **@Id** (ligne 9)

```

1 package entites;
2
3 import javax.persistence.Entity;
4 import javax.persistence.Id;
5
6 @Entity
7 public class Pole {
8
9     @Id
10     private String codePole;
11     private String nomPole;
12
13 ...

```

On accepte les 2 autres suggestions et on obtient en définitive:

Ont été ajoutés: un **constructeur vide** (ligne 14) et **implements Serializable** (ligne 8)

```

1 package entites;
2
3 import java.io.Serializable;
4 import javax.persistence.Entity;
5 import javax.persistence.Id;
6
7 @Entity
8 public class Pole implements Serializable {
9
10     @Id
11     private String codePole;
12     private String nomPole;
13
14     public Pole() {
15     }
16
17 ...

```


On applique les mêmes opérations à la classe **Salarie** et on obtient:

```

1 package entites;
2
3 import java.io.Serializable;
4 import javax.persistence.Entity;
5 import javax.persistence.Id;
6
7 @Entity
8 public class Salarie implements Serializable {
9
10     @Id
11     private Long      numsal;
12     private String    nomsal;
13     private String    sexe;
14     private Float     salaire;
15
16     public Salarie() {
17     }
18
19 ....

```

Si on refait un **Clean and Build** l'IDE nous ne fait plus de suggestions.

Nous allons passer à l'implémentation des attributs navigationnels.

11° Ajout d'une liste de salariés dans la classe Pole.

private List<Salarie> lesSalaries=new LinkedList<Salarie>();

```

1 package entites;
2
3 import java.io.Serializable;
4 import javax.persistence.Entity;
5 import javax.persistence.Id;
6
7 @Entity
8 public class Pole implements Serializable {
9
10     @Id
11     private String    codePole;
12     private String    nomPole;
13
14     private List<Salarie> lesSalaries=new LinkedList<Salarie>();
15
16     public Pole() {
17     }
18
19 ....

```

En cliquant sur l'ampoule l'IDE nous suggère:

```

12     private String    nomPole;
13
14     private List<Salarie> lesSalaries=new LinkedList<Salarie>();
15
16     Add import for java.util.LinkedList
17     Add import for java.util.List
18     Add import for java.awt.List
19     Add import for org.eclipse.persistence.internal.oxm.schema.model.List
20     Add import for com.sun.tools.javac.util.List
21     Add import for com.sun.xml.internal.bind.v2.schemagen.xmlschema.List
22     Create class "List" in package entites
23     Create class "List" in entites.Pole
24     Create class "LinkedList" in package entites
25     Create class "LinkedList" in entites.Pole
26
27     System.out.print(codePole+ " ");

```

Accepter les 2 premières propositions pour obtenir les **imports** suivants:

```

1 package entites;
2
3 import java.io.Serializable;
4 import java.util.LinkedList;
5 import java.util.List;
6 import javax.persistence.Entity;
7 import javax.persistence.Id;
8
9 @Entity
10 public class Pole implements Serializable {
11
12     @Id
13     private String    codePole;
14     private String    nomPole;
15
16     private List<Salarie> lesSalaries=new LinkedList<Salarie>();
17
18     public Pole() {
19     }
20
21 ....

```


L'ampoule est encore allumée, voyons ce qu'on nous propose en cliquant dessus:
On obtient:

```

1 package entites;
2
3 import java.io.Serializable;
4 import java.util.LinkedList;
5 import java.util.List;
6 import javax.persistence.Entity;
7 import javax.persistence.Id;
8
9 @Entity
10 public class Pole implements Serializable {
11
12     @Id
13     private String codePole;
14     private String nomPole;
15
16     private List<Salarie> lesSalaries=new LinkedList<Salarie>();
17
18     Create unidirectional OneToMany relationship
19     Create bidirectional OneToMany relationship...
20     Create bidirectional ManyToMany relationship...
21
22     public Pole(String codefonct, String libfonct) {

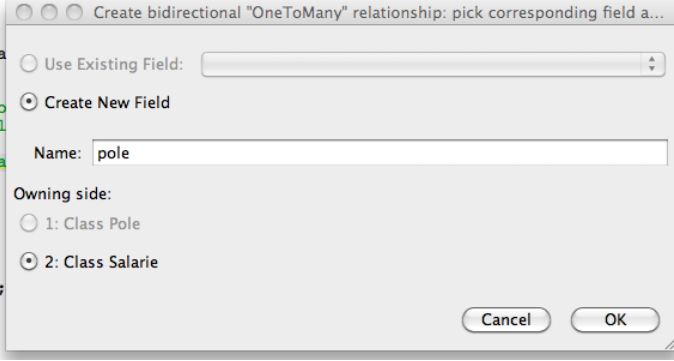
```

Nous choisissons un lien bidirectionnel OneToMany , on obtient alors:

```

1 package entites;
2
3 import java.io.Serializable;
4 import java.util.LinkedList;
5 import java.util.List;
6 import javax.persistence.Entity;
7 import javax.persistence.Id;
8
9 @Entity
10 public class Pole implements Seriala
11
12     @Id
13     private String codePo
14     private String nomPol
15
16     private List<Salarie> lesSala
17
18     public Pole() {
19     }
20
21     public Pole(String codefonct,
22
23         this.codePole = codefonct;
24         this.nomPole = libfonct;
25     }
26
27     public void afficher() {

```



Nous souhaitons que l'attribut dans la classe Salarie permettant de faire référence à la la classe **Pole** se nomme **lePole** donc, on change Name en lePole et après validation on obtient :

```

1 package entites;
2
3 import ...
4
5 @Entity
6 public class Pole implements Serializable {
7
8     @Id
9     private String codePole;
10     private String nomPole;
11
12     @OneToMany(mappedBy = "lePole")
13     private List<Salarie> lesSalaries=new LinkedList<Salarie>();
14
15     public Pole() {
16     }
17
18     public Pole(String codefonct, String libfonct) {
19
20         this.codePole = codefonct;
21         this.nomPole = libfonct;
22     }
23
24     public void afficher(){
25
26         System.out.print(codePole+ " ");
27         System.out.print(nomPole);
28     }
29
30     Getters et setters
31
32 }

```

Il s'est aussi passé aussi quelque chose dans la classe **Salarie**.

Ouvrir le code de **Salarie**:

```

1 package entites;
2
3 import java.io.Serializable;
4 import javax.persistence.Entity;
5 import javax.persistence.Id;
6 import javax.persistence.ManyToOne;
7
8 @Entity
9 public class Salarie implements Serializable {
10
11     @Id
12     private Long numsal;
13     private String nomsal;
14     private String sexe;
15     private Float salaire;
16
17     @ManyToOne
18     private Pole lePole;
19
20     public Salarie() {
21     }
22
23 ...

```

Les lignes 17 et 18 ont été ajoutées:

@ManyToOne
private Pole le Pole;

Nous ajoutons l'annotation **@JoinColumn(name="CODEPOLE")** dans la classe **Salarie**, avant l'annotation **@ManyToOne**. (Cela permet d'obtenir dans la table **Salarie** qui va être créée une clé étrangère vers la table **Pole** de nom **CODEPOLE**)

Et après avoir accepté l'importation proposée, nous obtenons le code suivant:

```

1 package entites;
2
3 import java.io.Serializable;
4 import javax.persistence.Entity;
5 import javax.persistence.Id;
6 import javax.persistence.JoinColumn;
7 import javax.persistence.ManyToOne;
8
9 @Entity
10 public class Salarie implements Serializable {
11
12     @Id
13     private Long numsal;
14     private String nomsal;
15     private String sexe;
16     private Float salaire;
17
18     @JoinColumn(name="CODEPOLE")
19     @ManyToOne
20     private Pole lePole;
21
22     public Salarie() {
23
24 ...

```

12) Nous allons ajouter les getters et setters pour l'attribut **lePole** de la classe **Salarie** et l'attribut **lesSalaries** de la classe **Pole**.

clique droit et choisir **Insert Code** puis **Getters and Setters**

Dans le code de la classe **Salarie** vous devez obtenir:

```

43
44 Getters et setters
45
46 public Pole getLePole() {
47     return lePole;
48 }
49
50 public void setLePole(Pole lePole) {
51     this.lePole = lePole;
52 }
53
54 }
55
56 ...

```

Et dans le code de la classe **Pole** vous devez obtenir:

```

35
36 Getters et setters
37
38 public List<Salarie> getLesSalaries() {
39     return lesSalaries;
40 }
41
42 public void setLesSalaries(List<Salarie> lesSalaries) {
43     this.lesSalaries = lesSalaries;
44 }
45
46 }
47
48 ...

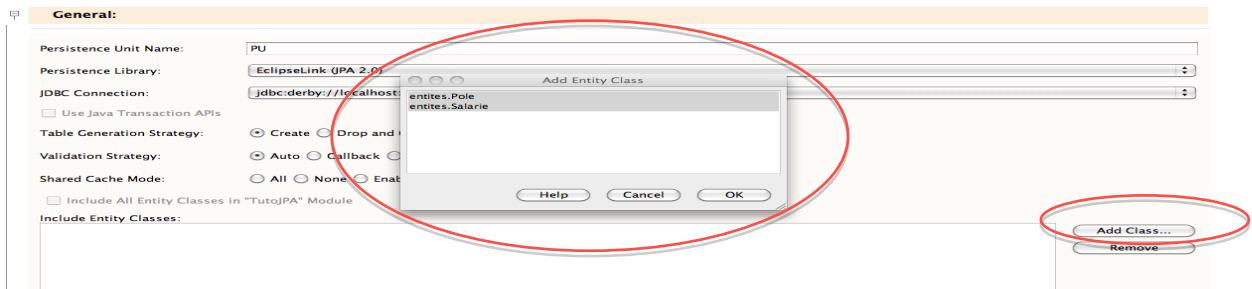
```

Vous rangerez ces getters et setters dans leurs folders.

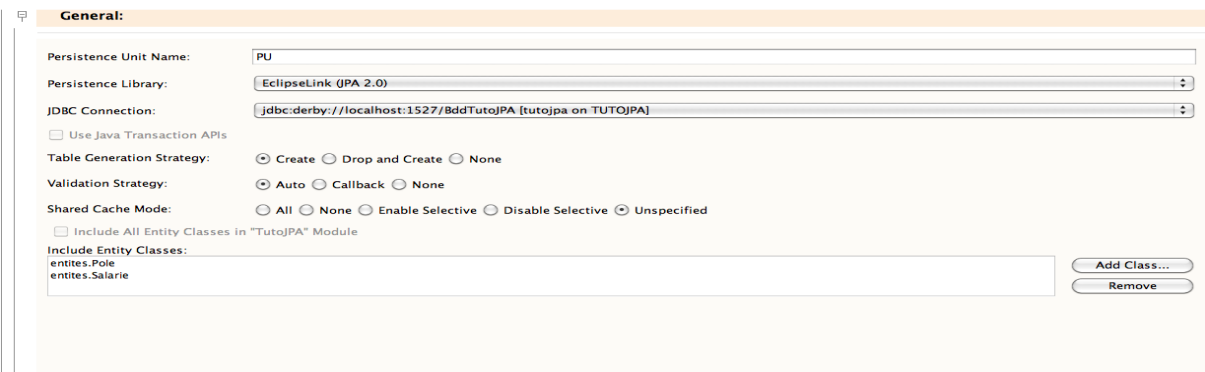
13) Fin du paramétrage du fichier Persistence.xml.

Il faut maintenant indiquer dans le fichier Persistence.xml (Persistence Unit) que l'on souhaite que les classes Fonction et Salarie soient gérées par le framework de persistance.

Donc dans **Persistence Unit** , en mode Design, on sélectionne ces deux classes en utilisant les bouton **Add Class...**



Sélectionner les deux entités **Pole** et **Salarie** pour obtenir:



14) Création des tables de la base de données.

Nous allons maintenant lancer la classe **CreerBase** du package **tutojpa**. Cette permet de créer la base qui contiendra les tables correspondant à ces deux classes:

Le code en est actuellement le suivant:

```

1 package tutojpa;
2
3
4 public class CreerBase {
5
6     public static void main(String[] args) {
7
8         // EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
9         // EntityManager em = emf.createEntityManager();
10
11     }
12 }
13

```

Décommenter les lignes 9 et 10 pour obtenir:

```

1 package tutojpa;
2
3
4 public class CreerBase {
5
6     public static void main(String[] args) {
7
8         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
9         EntityManager em = emf.createEntityManager();
10
11     }
12
13 }

```

Accepter les imports au fur et à mesure qu'il sont suggérés par l'IDE, afin d'obtenir ceci:

```

1 package tutojpa;
2
3
4 import javax.persistence.EntityManager;
5 import javax.persistence.EntityManagerFactory;
6 import javax.persistence.Persistence;
7
8 public class CreerBase {
9
10     public static void main(String[] args) {
11
12         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
13         EntityManager em = emf.createEntityManager();
14
15     }
16
17 }

```

Au préalable paramétrer les propriétés de Persistence.xml comme suit:

(utiliser le bouton Add dans propriétés en bas de la page et chercher dans la longue liste qui apparaît alors)

☐ Use Java Transaction APIs

Table Generation Strategy: ☐ Create ☐ Drop and Create ☒ None

Validation Strategy: ☒ Auto ☐ Callback ☐ None

Shared Cache Mode: ☐ All ☐ None ☐ Enable Selective ☐ Disable Selective ☒ Unspecified

☐ Include All Entity Classes in "TutoJPA" Module

Include Entity Classes:

entites.Pole
entites.Salarie

Add Class... Remove

Properties:

Name	Value
eclipselink.create-ddl-jdbc-file-name	commandesSQLCreation
eclipselink.ddl-generation.output-mode	both

Add... Edit... Remove

Le fichier **commandesSQLCreation** contiendra les **ordres SQL de création de tables** que nous allons générer automatiquement grâce au **gestionnaire de persistance**.

Clean and Build

Puis exécuter **CreerBase**, on obtient alors dans la console:

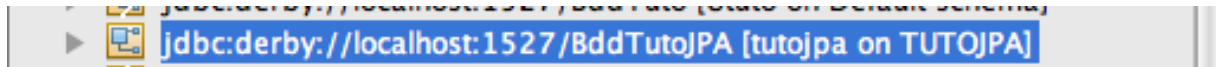
```

Java DB Database Process * TutoJPA (run)
[EL Info]: 2015-03-13 19:33:19.424--ServerSession(822056113)--EclipseLink, version: Eclipse Persistence Services - 2.
[EL Info]: 2015-03-13 19:33:19.807--ServerSession(822056113)--file:/Users/rsmon/Nb7Proj/TutoJPA/build/classes/_PU log
BUILD SUCCESSFUL (total time: 2 seconds)

```

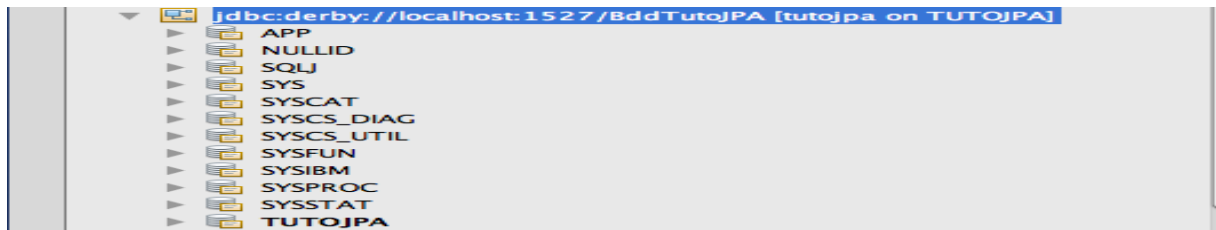
15) Vérification: Les tables ont bien été créées.

Allons voir maintenant dans la base de données **BddTutoJPA** (Onglet Services puis DataBases):

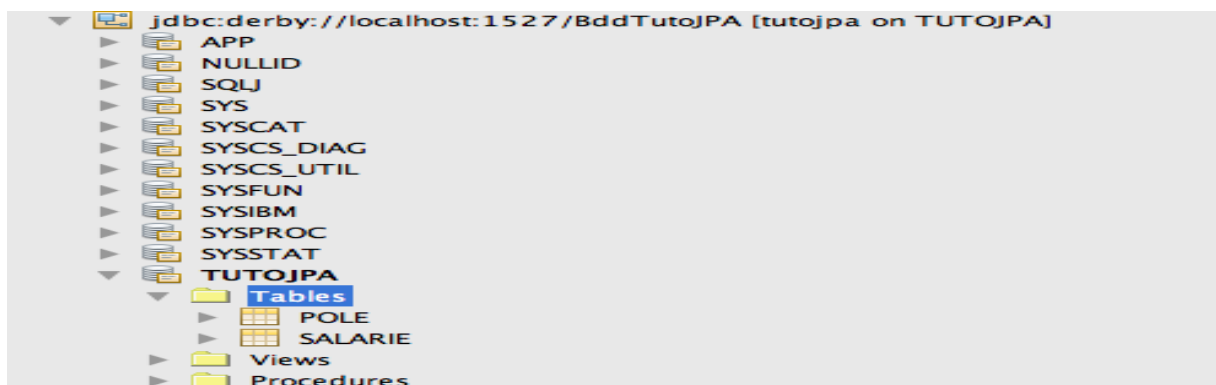


Cliquer droit et sélectionner connect dans le popup, puis double cliquer sur la ligne représentée ci-dessus.

On obtient alors:

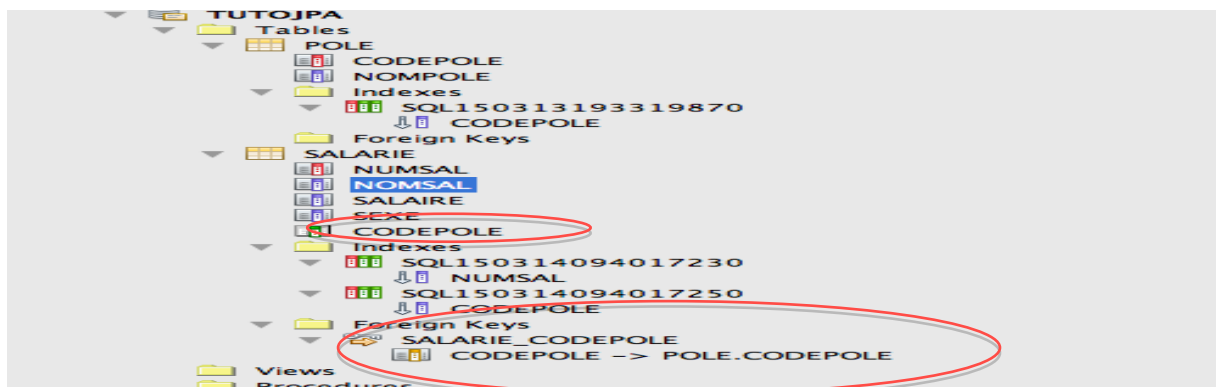


On ouvre l'item **TUTOJPA**, puis **Tables** une fois l'item TUTOJPA déployé. On obtient alors:



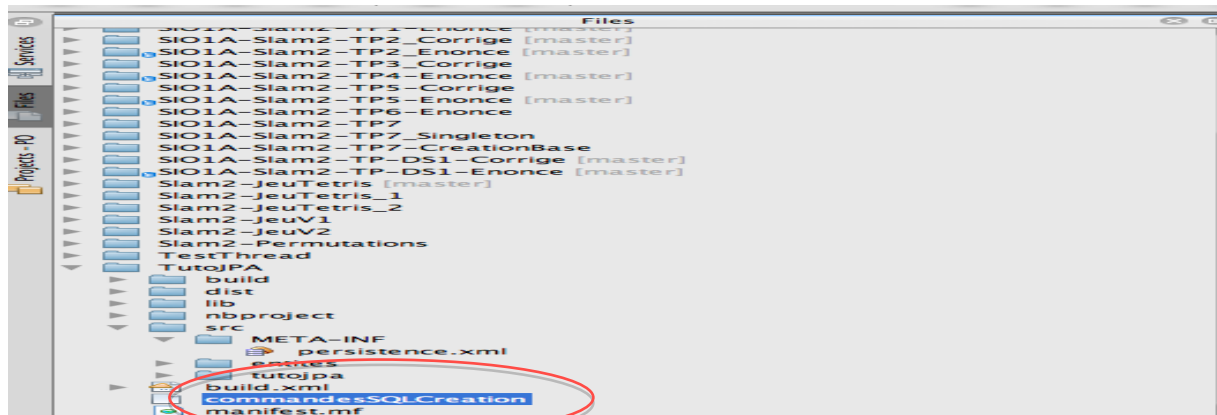
Les deux tables ont bien été créées.

En déployant les items FONCTION et SALARIE nous obtenons:



Dans la table SALARIE, nous avons la clé étrangère CODEPOLE.

Nous pouvons observer les commandes de création de tables qui ont été générées par le **framework de persistance** dans la perspective **Files** du projet:



NB: On fait apparaître la perspective **File** en sélectionnant **Windows** puis **Files** dans la Barre de Menu.

Voici le contenu du fichier **CommandesSQLCreation** (remis en forme pour faciliter la lecture).

```

1 CREATE TABLE POLE
2 (
3     CODEPOLE VARCHAR(255) NOT NULL,
4     NOMPOLE VARCHAR(255),
5     PRIMARY KEY (CODEPOLE)
6 )
7
8 CREATE TABLE SALARIE
9 (
10     NUMSAL BIGINT NOT NULL,
11     NOMSAL VARCHAR(255),
12     SALAIRE FLOAT,
13     SEXE VARCHAR(255),
14     CODEPOLE VARCHAR(255),
15     PRIMARY KEY (NUMSAL))
16
17 ALTER TABLE SALARIE ADD CONSTRAINT SALARIE_CODEPOLE FOREIGN KEY (CODEPOLE) REFERENCES POLE (CODEPOLE)

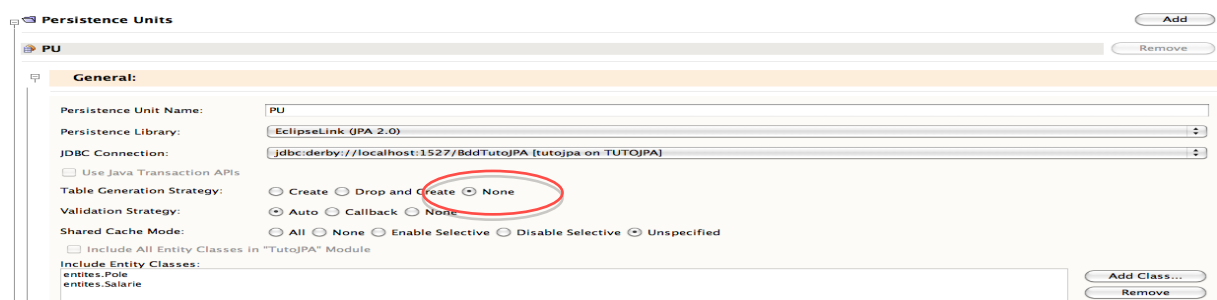
```

La Base étant créée, nous allons pouvoir y enregistrer des données, les consulter, les modifier ou les supprimer en utilisant l'**API JPA** (**JPA = Java Persistence API**).

16) Modification du paramétrage du Gestionnaire de persistance.

Avant tout, il faut indiquer au gestionnaire de persistance qu'il ne faut plus créer les tables, puisqu'elles existent.

Pour cela ouvrir **Persistence Unit** et positionner **Table Generation Strategy** à **None**.



Recompiler le projet: **Clean and Build** .

17) Création d'un Jeu d'essais.

Nous allons pour cela exécuter la classe **ChargerBase** présentée ci-dessous.

Il faut d'abord décommenter les lignes et faire les imports nécessaires (choisir **source** dans la barre de menu et sélectionner **Fixe Imports**) pour obtenir le code suivant:

```

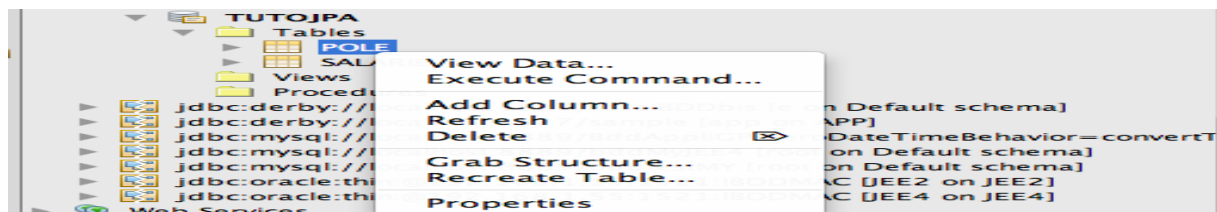
1 package tutojpa;
2
3 import entites.Pole;
4 import entites.Salarie;
5 import javax.persistence.EntityManager;
6 import javax.persistence.EntityManagerFactory;
7 import javax.persistence.Persistence;
8
9 public class ChargerBase {
10
11     public static void main(String[] args) {
12
13         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
14         EntityManager em = emf.createEntityManager();
15
16         Pole p1=new Pole("Slam","Solutions Logicielles et Applications Métier");
17         Pole p2=new Pole("Slsr","Solutions d'Infrastructures Systèmes et Réseaux");
18
19         Salarie s1=new Salarie(101L,"Dupont Pierre","M", 1750F);
20         Salarie s2=new Salarie(102L,"Martin Sophie","F", 1900F);
21         Salarie s3=new Salarie(103L,"Deruelle Marc","M", 1655F);
22         Salarie s4=new Salarie(104L,"Leleu Anne","F", 1900F);
23
24         s1.setPole(p1);
25         s2.setPole(p1);
26         s3.setPole(p2);
27         s4.setPole(p2);
28
29         em.getTransaction().begin();
30
31         em.persist(p1);em.persist(p2);
32
33         em.persist(s1);em.persist(s2);
34         em.persist(s3);em.persist(s4);
35
36         em.getTransaction().commit();
37     }
38 }

```

Exécuter cette classe.

Vérification dans la base:

POLE: clique droit sur Pole et View Data



On obtient:

#	CODEPOLE	NOMPOLE
1	Slam	Solutions Logicielles et Applications Métier
2	Slsr	Solutions d'Infrastructures Systèmes et Réseaux

Pour SALARIE on obtient:

#	NUMSAL	NOMSAL	SALAIRE	SEXE	CODEPOLE
1	102	Martin Sophie	1900.0 F		Slam
2	103	Deruelle Marc	1655.0 M		Slsr
3	101	Dupont Pierre	1750.0 M		Slam
4	104	Leleu Anne	1900.0 F		Slsr

16) Consultation de la base

Créer exécuter les Java Main Classes suivantes:

Consultation1:

```

1  package tutojpa;
2
3
4  import entites.Pole;
5  import javax.persistence.EntityManager;
6  import javax.persistence.EntityManagerFactory;
7  import javax.persistence.Persistence;
8
9  public class Consultation01 {
10
11     public static void main(String[] args) {
12
13         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
14         EntityManager        em = emf.createEntityManager();
15
16         // Recherche du Pôle de code Slam et affichage
17
18         Pole pole = em.find(Pole.class, "Slam");
19
20         System.out.println();
21
22         pole.afficher();
23
24         System.out.println("\n");
25
26     }
27 }

```

Output - TutoJPA (run)

```

[EL Info]: 2015-03-14 11:22:23.533--ServerSession(1268020374)--EclipseLink, version: Eclipse Persistence Servi
[EL Info]: 2015-03-14 11:22:23.901--ServerSession(1268020374)--file:/Users/rsmon/Nb7Proj/TutoJPA/build/classes
Slam Solutions Logicielles et Applications Métier

```

Consultation2:

```

1  package tutojpa;
2
3
4  import entites.Pole;
5  import entites.Salarie;
6  import javax.persistence.EntityManager;
7  import javax.persistence.EntityManagerFactory;
8  import javax.persistence.Persistence;
9
10 public class Consultation02 {
11
12     public static void main(String[] args) {
13
14         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
15         EntityManager        em = emf.createEntityManager();
16
17         // Recherche du Pôle de code Slam et affichage du nom du Pôle et de la liste des salariés du Pôle
18
19         Pole pole = em.find(Pole.class, "Slam");
20
21         System.out.println();
22
23         pole.afficher();
24
25         System.out.println("\n");
26
27         for (Salarie sal : pole.getLesSalaries()){
28             sal.afficher();
29             System.out.println();
30         }
31
32         System.out.println();
33     }
34 }
35

```

Output - TutoJPA (run)

```

[EL Info]: 2015-03-14 11:15:56.262--ServerSession(2106232034)--EclipseLink, version: Eclipse Persistence S
[EL Info]: 2015-03-14 11:15:56.71--ServerSession(2106232034)--file:/Users/rsmon/Nb7Proj/TutoJPA/build/clas
Slam Solutions Logicielles et Applications Métier

```

```

102 Martin Sophie F 1900.0
101 Dupont Pierre M 1750.0

```

Consultation3:

```

1 package tutojpa;
2
3
4 import entites.Salarie;
5 import javax.persistence.EntityManager;
6 import javax.persistence.EntityManagerFactory;
7 import javax.persistence.Persistence;
8
9 public class Consultation03 {
10
11     public static void main(String[] args) {
12
13         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
14         EntityManager em = emf.createEntityManager();
15
16         // Recherche du Salarie de numero 103, affichage des ses information et du Pôle auquel il appartient
17
18         System.out.println();
19
20         Salarie sal= em.find(Salarie.class, 103L);
21
22         sal.afficher();
23
24         System.out.println("\n");
25
26         System.out.print (sal.getLePole().getCodePole()+ " ");
27
28         System.out.println(sal.getLePole().getNomPole());
29
30         System.out.println();
31     }
32 }

```

Output - TutoJPA (run)

```

[EL Info]: 2015-03-14 11:27:40.195--ServerSession(2058536614)--EclipseLink, version: Eclipse Persistence Servi
[EL Info]: 2015-03-14 11:27:40.576--ServerSession(2058536614)--file:/Users/rsmon/Nb7Proj/TutoJPA/build/classes
103 Deruelle Marc M 1655.0
Sisr Solutions d'Infrastructures Systèmes et Réseaux

```

Consultation 4:

```

1 package tutojpa;
2
3
4 import entites.Salarie;
5 import java.util.List;
6 import javax.persistence.EntityManager;
7 import javax.persistence.EntityManagerFactory;
8 import javax.persistence.Persistence;
9
10 public class Consultation04 {
11
12     public static void main(String[] args) {
13
14         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
15         EntityManager em = emf.createEntityManager();
16
17         // Affichage de tous les salariés et du Pôle auquel ils appartiennent
18
19         System.out.println();
20
21         List<Salarie> lesSalaries= em.createQuery("Select sal From Salarie sal").getResultList();
22
23         for (Salarie sal: lesSalaries){
24
25             sal.afficher();
26
27             System.out.print ( " " + sal.getLePole().getCodePole()+ " ");
28             System.out.println( sal.getLePole().getNomPole());
29         }
30
31         System.out.println();
32     }
33 }
34 }

```

Output - TutoJPA (run)

```

[EL Info]: 2015-03-14 11:37:17.375--ServerSession(1118841128)--EclipseLink, version: Eclipse Persist
[EL Info]: 2015-03-14 11:37:17.836--ServerSession(1118841128)--file:/Users/rsmon/Nb7Proj/TutoJPA/bui
102 Martin Sophie F 1900.0 Slam Solutions Logicielles et Applications Métier
103 Deruelle Marc M 1655.0 Sisr Solutions d'Infrastructures Systèmes et Réseaux
101 Dupont Pierre M 1750.0 Slam Solutions Logicielles et Applications Métier
104 Leleu Anne F 1900.0 Sisr Solutions d'Infrastructures Systèmes et Réseaux

```

Consultation 5:

```

1 package tutojpa;
2
3 import entites.Pole;
4 import entites.Salarie;
5 import java.util.List;
6 import javax.persistence.EntityManager;
7 import javax.persistence.EntityManagerFactory;
8 import javax.persistence.Persistence;
9
10 public class Consultation05 {
11
12     public static void main(String[] args) {
13
14         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
15         EntityManager em = emf.createEntityManager();
16
17         // Affichage de tous les Pôles avec leurs salariés
18
19         System.out.println();
20
21         List<Pole> lesPoles = em.createQuery("Select pole From Pole pole").getResultList();
22
23         for (Pole p : lesPoles){
24
25             p.afficher(); System.out.println("\n");
26
27             for (Salarie sal : p.getLesSalaries()){
28
29                 System.out.print(" ");
30                 sal.afficher();
31                 System.out.println();
32             }
33
34             System.out.println();
35         }
36
37         System.out.println();
38     }
39 }

```

Output - TutoJPA (run)

[EL Info]: 2015-03-14 11:45:20.422--ServerSession(1190000432)--EclipseLink, version: Eclipse Per
 [EL Info]: 2015-03-14 11:45:20.782--ServerSession(1190000432)--file:/Users/rsmon/Nb7Proj/TutoJPA

Sisr Solutions d'Infrastructures Systèmes et Réseaux

103 Deruelle Marc M 1655.0
 104 Leleu Anne F 1900.0

Slam Solutions Logicielles et Applications Métier

102 Martin Sophie F 1900.0
 101 Dupont Pierre M 1750.0

17) Modifications dans la base:

a) Modification du salaire du salarie 101:

```

1 package tutojpa;
2
3 import entites.Salarie;
4 import javax.persistence.EntityManager;
5 import javax.persistence.EntityManagerFactory;
6 import javax.persistence.Persistence;
7
8 public class Modification01 {
9
10     public static void main(String[] args) {
11
12         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
13         EntityManager em = emf.createEntityManager();
14
15         // Modification du salaire du salarie de numero 101
16
17         Salarie sal = em.find(Salarie.class, 101L);
18
19         System.out.println();
20
21         sal.afficher();
22
23         System.out.println("\n");
24
25         em.getTransaction().begin();
26
27         sal.setSalaire(1830F);
28
29         em.getTransaction().commit();
30     }
31 }
32

```

Vérification dans la base:

Modification01.java | SQL Command 3

Source | History | Connection: jdbc:derby://localhost:1527/BddTutoJPA [tutojpa on TUT...]

1 select * from TUTOJPA.SALARIE;

2

select * from TUTOJPA.SALARIE

Page Size: 20 | Total Rows: 4 | Page: 1 of 1 | Matching Rows:

#	NUMSAL	NOMSAL	SALAIRE	SEXE	CODEPOLE
1		102 Martin Sophie		1900.0 F	Slam
2		103 Deruelle Marc		1655.0 M	Sisr
3		101 Dupont Pierre		1830.0 M	Slam
4		104 Leleu Anne		1900.0 F	Sisr

b) Augmentation de 5% de tous les salariés du Pôle de code Slam:

```

2 package tutojpa;
3
4 import entites.Pole;
5 import entites.Salarie;
6 import javax.persistence.EntityManager;
7 import javax.persistence.EntityManagerFactory;
8 import javax.persistence.Persistence;
9
10 public class Modification02 {
11
12     public static void main(String[] args) {
13
14         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
15         EntityManager em = emf.createEntityManager();
16
17         // Modification de 5% les salaires du Pôle Slam
18
19         Pole pole=em.find(Pole.class, "Slam");
20
21         em.getTransaction().begin();
22         for (Salarie sal : pole.getLesSalaries()){
23             sal.augmenter(5.0f);
24         }
25         em.getTransaction().commit();
26     }
27 }

```

The screenshot shows the Eclipse IDE with the SQL Command window open. The query is `select * from TUTOJPA.SALARIE;`. The results are displayed in a table with 6 columns: #, NUMSAL, NOMSAL, SALAIRE, SEXE, and CODEPOLE. The table contains 4 rows of data.

#	NUMSAL	NOMSAL	SALAIRE	SEXE	CODEPOLE
1	102	Martin Sophie	1994.9998779296875	F	Slam
2	103	Deruelle Marc	1655.0	M	Sisr
3	101	Dupont Pierre	1921.4998779296875	M	Slam
4	104	Leleu Anne	1900.0	F	Sisr

c) Suppression du salaire de numéro 104.

```

1 package tutojpa;
2
3
4 import entites.Salarie;
5 import javax.persistence.EntityManager;
6 import javax.persistence.EntityManagerFactory;
7 import javax.persistence.Persistence;
8
9 public class Modification03 {
10
11     public static void main(String[] args) {
12
13         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
14         EntityManager em = emf.createEntityManager();
15
16         // Suppression du salarié de numero 104
17
18         Salarie sal=em.find(Salarie.class, 104L);
19
20         em.getTransaction().begin();
21         em.remove(sal);
22         em.getTransaction().commit();
23     }
24 }

```

Vérification dans la base:

The screenshot shows the Eclipse IDE with the SQL Command window open. The query is `select * from TUTOJPA.SALARIE;`. The results are displayed in a table with 6 columns: #, NUMSAL, NOMSAL, SALAIRE, SEXE, and CODEPOLE. The table now contains 3 rows of data, as record 104 has been deleted.

#	NUMSAL	NOMSAL	SALAIRE	SEXE	CODEPOLE
1	102	Martin Sophie	1994.9998779296875	F	Slam
2	103	Deruelle Marc	1655.0	M	Sisr
3	101	Dupont Pierre	1921.4998779296875	M	Slam

d) Création des Salariés suivants:

110 Legrain Daniel M 1814 dans le Pôle Slam
 120 Duchemin Léa F 1920 dans le Pôle Sisir

```

1 package tutojpa;
2
3
4 import entites.Pole;
5 import entites.Salarie;
6 import javax.persistence.EntityManager;
7 import javax.persistence.EntityManagerFactory;
8 import javax.persistence.Persistence;
9
10 public class Modification04 {
11
12     public static void main(String[] args) {
13
14         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
15         EntityManager em = emf.createEntityManager();
16
17         // Ajout de 2 salariés
18
19         Pole pole1=em.find(Pole.class,"Slam");
20         Salarie sal1=new Salarie(110L,"Legrain Daniel","M", 1814f);
21         sal1.setLePole(pole1);
22
23         Pole pole2=em.find(Pole.class,"Sisir");
24         Salarie sal2=new Salarie(120L,"Duchemin Léa", "F", 1920f);
25         sal2.setLePole(pole2);
26
27         em.getTransaction().begin();
28         em.persist(sal1);em.persist(sal2);
29
30         em.getTransaction().commit();
31     }
32 }
33

```

Vérification dans la base:

The screenshot shows the Eclipse IDE with a SQL Command window. The query 'select * from TUTOJPA.SALARIE;' is entered. The results are displayed in a table with 5 rows and 6 columns: #, NUMSAL, NOMSAL, SALAIRE, SEXE, and CODEPOLE.

#	NUMSAL	NOMSAL	SALAIRE	SEXE	CODEPOLE
1	102	Martin Sophie	1994.9998779296875	F	Slam
2	103	Deruelle Marc	1655.0	M	Sisir
3	101	Dupont Pierre	1921.4998779296875	M	Slam
4	120	Duchemin Léa	1920.0	F	Sisir
5	110	Legrain Daniel	1814.0	M	Slam

Relançons la classe **Consultation05**, on obtient :

The screenshot shows the Eclipse IDE output window. The output displays the results of the Consultation05 class, showing the names of the poles and the list of employees for each pole.

```

Output - TutoJPA (run)
[EL Info]: 2015-03-14 12:35:49.371--ServerSession(1783079124)--EclipseLink, v
[EL Info]: 2015-03-14 12:35:49.826--ServerSession(1783079124)--file:/Users/rs

Sisir Solutions d'Infrastructures Systèmes et Réseaux
103 Deruelle Marc M 1655.0
120 Duchemin Léa F 1920.0

Slam Solutions Logicielles et Applications Métier
102 Martin Sophie F 1994.9999
101 Dupont Pierre M 1921.4999
110 Legrain Daniel M 1814.0

```

e) Finalement salarié de numéro 110 qui est actuellement au Pôle **Slam** est maintenant affecté au Pôle **Sisr**.

```

1 package tutojpa;
2
3
4 import entites.Pole;
5 import entites.Salarie;
6 import javax.persistence.EntityManager;
7 import javax.persistence.EntityManagerFactory;
8 import javax.persistence.Persistence;
9
10 public class Modification05 {
11
12     public static void main(String[] args) {
13
14         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PU");
15         EntityManager em = emf.createEntityManager();
16
17         // Le Salarie 110 passe du Pôle Slam au Pôle Sisr
18
19
20         Salarie sal= em.find(Salarie.class, 110L);
21
22         Pole pole=em.find(Pole.class, "Sisr");
23
24         em.getTransaction().begin();
25
26         sal.setLePole(pole);
27
28         em.getTransaction().commit();
29     }
30 }

```

Vérification dans la base:

#	NUMSAL	NOMSAL	SALAIRE	SEXE	CODEPOLE
1	102	Martin Sophie	1994.99987792968...	F	Slam
2	103	Deruelle Marc	1655.0	M	Sisr
3	101	Dupont Pierre	1921.49987792968...	M	Slam
4	120	Duchemin Léa	1920.0	F	Sisr
5	110	Legrain Daniel	1814.0	M	Sisr

Réexécutons la classe **Consultation05** et on obtient:

```

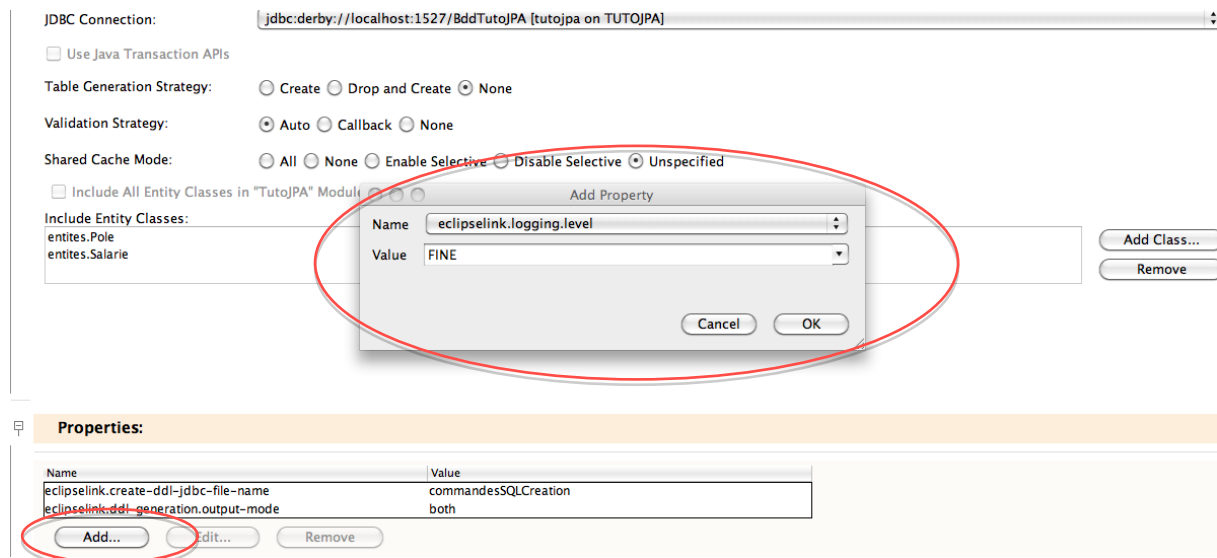
Sisr Solutions d'Infrastructures Systèmes et Réseaux
103 Deruelle Marc M 1655.0
120 Duchemin Léa F 1920.0
110 Legrain Daniel M 1814.0

Slam Solutions Logicielles et Applications Métier
102 Martin Sophie F 1994.9999
101 Dupont Pierre M 1921.4999

```

18) Paramétrage de Persistence.xml pour voir les requêtes SQL exécutées:

Dans properties cliquer sur Add et faire la sélection indiquée ci-dessous:



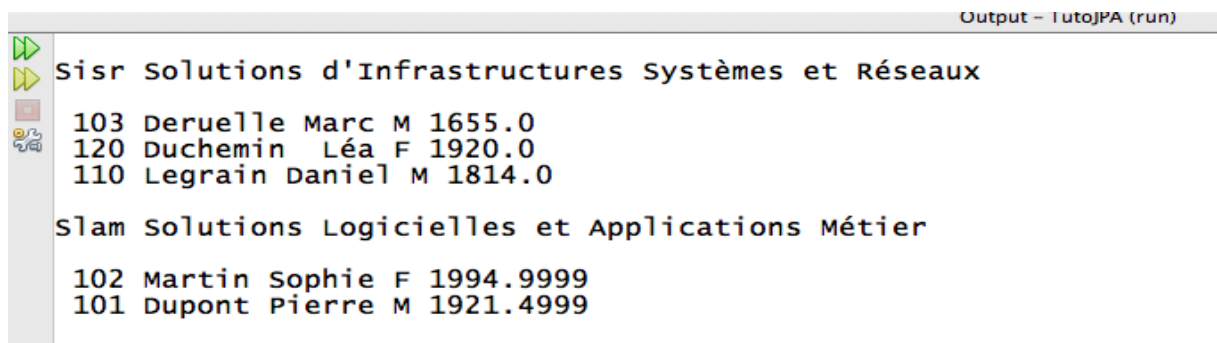
Refaire un build.

Relancez par exemple **Consultation05** et **observez les affichages dans la console**, vous y trouverez les **requêtes SQL exécutées par le gestionnaire de persistance EclipseLink**.

NB: L'affichage obtenu n'a pas été intégré dans ce document car trop volumineux.

On peut aussi demander de n'avoir aucun message d'EclipseLink en positionnant le paramètre **eclipselink.loggin.level** à **OFF**.

Si on réexécute **Consultation05**, on a alors:



Il n'y a plus que le résultat qui est affiché, les informations du gestionnaire de persistance ne sont plus affichées.