Introduction à JPA
(Java Persistence API)

Fouomene Pewo Daniel Rene

FREELANCERTECH

Une équipe d'experts seniors dynamique à votre service...

Une équipe d'experts dans leur domaine technologique qui ont décidé de se mettre ensemble pour offrir leurs services en fonction de leurs disponibilités,

www.freelancertech.net

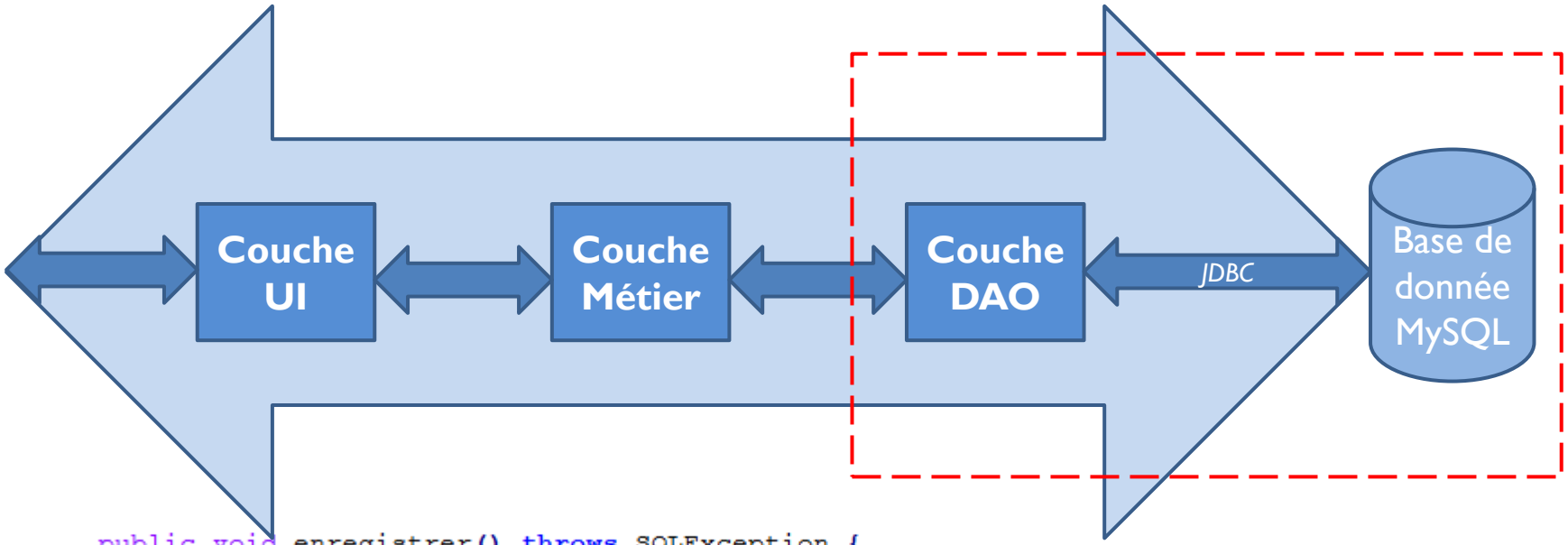
www.facebook.com/pages/FreelancerTech/822357474482862

www.twitter.com/FREELANCERTECH (@FREELANCERTECH)

Plan

- Historique
- ORM : Object Relational Mapping
- JPA : Java Persistence Api
- Entity: Entités
- Autres annotations
- Contexte de persistance
- Opérations prises en charge par le gestionnaire d'entités
- Cycle de vie d'une instance d'entité
- Obtention d'une fabrique EntityManagerFactory
- Création du gestionnaire d'entité EntityManager
- Exemple d'insertion d'un livre
- Relations entre entités
- Référence
- Questions

Historique : Accès directement à la base de donnée grâce à l'API standard JDBC de Java




```
public void enregistrer() throws SQLException {
        //ouverture de la connexion jdbc
        Class.forName("com.mysql.jdbc.driver");
        String url = "jdbc:mysql://localhost/Bdbanque";
        Connection con = DriverManager.getConnection(url, "login", "password");

        //preparation  ou construction de la requête sql
        String insertStatement = "Insert into Client(Nom, Prenom,Nature) values (?, ?, ?)";
        PreparedStatement prepStm1 = con.prepareStatement(insertStatement);
        prepStm1.setString(1,txtNom.getText() );
        prepStm1.setString(2,txtPrenom.getText());
        prepStm1.setString(3,txtAge.getText() );
        //execution de la requête
        prepStm1.executeUpdate();
        prepStm1.close();

        //fermeture de la connexion jdbc
        con.close();
}
```

FREELANCERTECH

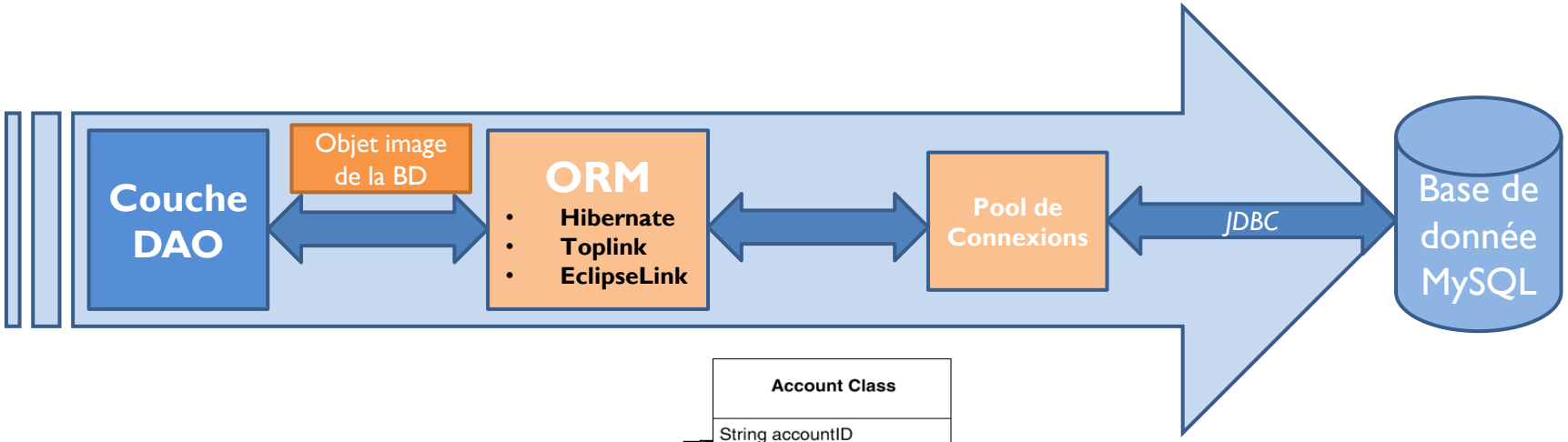
Problématiques de l'accès direct à la BD à l'aide de jdbc :

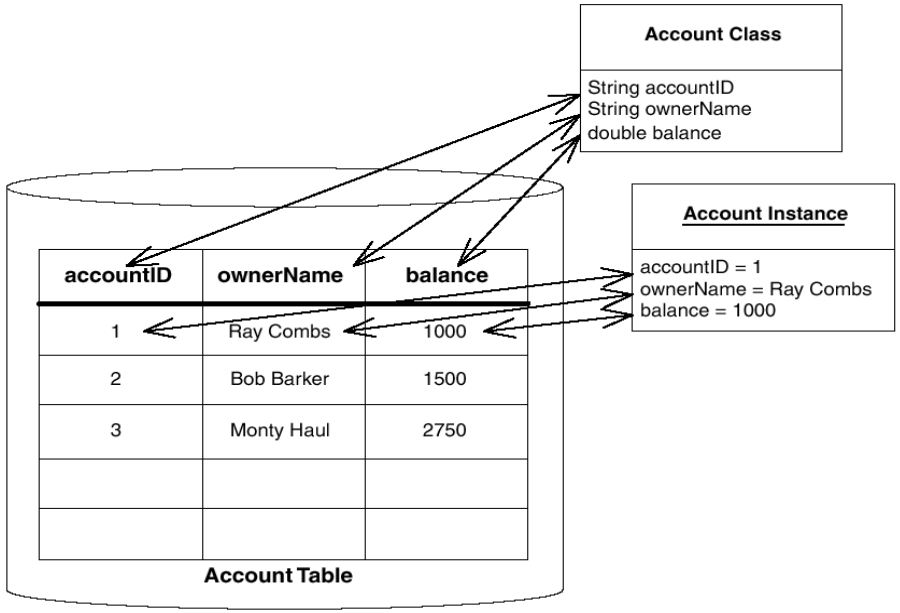
- Pour des raisons de performances, le coût d'ouverture / fermeture d'une connexion n'est pas négligeable,
  - Connection con = DriverManager.getConnection(url, "login", "password");
  - //les ordres SQL
  - con.close();
- L'utilisation du langage SQL rend la couche DAO difficilement maintenable,
  - String insertStatement = "Insert into Client(Nom, Prenom,Nature) values (?, ?, ?)"

Pour pallier à cette problématique et faciliter l'écriture de la couche DAO,

La communauté Java a donc fait naître des **Frameworks ORM** , tels que **Hibernate**, **Toplink**, **EclipseLink**

ORM : Object Relational Mapping





[1]

ReFactoring de la fonction enregistrée de notre couche DAO

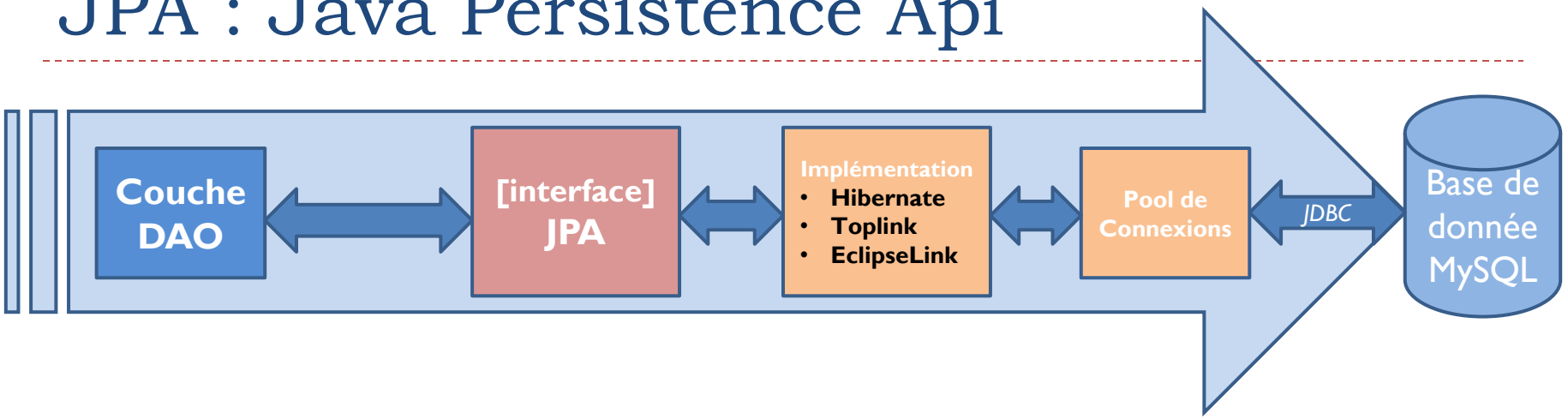
```
public void enregistrer() throws SQLException {
        //ouverture de la connexion jdbc
        Class.forName("com.mysql.jdbc.driver");
        String url = "jdbc:mysql://localhost/Bdbanque";
        Connection con = DriverManager.getConnection(url, "login", "password");

        //preparation ou construction de la requête sql
        String insertStatement = "Insert into Client(Nom, Prenom,Nature) values (?, ?, ?)";
        PreparedStatement prepStm1 = con.prepareStatement(insertStatement);
        prepStm1.setString(1,txtNom.getText() );
        prepStm1.setString(2,txtPrenom.getText());
        prepStm1.setString(3,txtAge.getText() );
        //execution de la requête
        prepStm1.executeUpdate();
        prepStm1.close();

        //fermeture de la connexion jdbc
        con.close();
}
```

```
// Avec l'ORM Hibernate
public void enregistrer() throws SQLException {

    Client client = new Client( txtNom.getText(),txtPrenom.getText(),txtAge.getText());
    HibernateManager orm = HibernateManager.getORM();
    orm.saveHibernate(client);

}
// Avec L'ORM TopLink
public void enregistrer() throws SQLException {

    Client client = new Client( txtNom.getText(),txtPrenom.getText(),txtAge.getText());
    TopLinkManager orm = TopLinkManager.getORM();
    orm.saveTopLink(client);

}
```

Devant le succès des Frameworks ORM,

Sun a décidé de standardiser une couche ORM via une spécification appelée **JPA** apparue en même temps que Java 5

JPA : Java Persistence Api



La spécification JPA est un ensemble d'interface du package javax.persistence

Exemple :

javax.persistence.Entity;
javax.persistence.EntityManagerFactory;
javax.persistence.EntityManager;
javax.persistence.EntityTransaction;

[**EntityManager**]
void **persist(**Entity entité**)**

Refactoring de la fonction enregistrée de notre couche DAO

```
// Avec l'ORM Hibernate
public void enregistrer() throws SQLException {

    Client client = new Client( txtNom.getText(),txtPrenom.getText(),txtAge.getText());
    HibernateManager orm = HibernateManager.getORM();
    orm.saveHibernate(client);

}

// Avec L'ORM TopLink
public void enregistrer() throws SQLException {

    Client client = new Client( txtNom.getText(),txtPrenom.getText(),txtAge.getText());
    TopLinkManager orm = TopLinkManager.getORM();
    orm.saveTopLink(client);

}


// Avec JPA
EntityManagerFactory emf = Persistence.createEntityManagerFactory( "JpaPU");
EntityManager em = emf.createEntityManager();

public void enregistrer() throws SQLException {
    Client client = new Client( txtNom.getText(),txtPrenom.getText(),txtAge.getText());
    em.persist(client)

}
```

JPA : Java Persistence Api

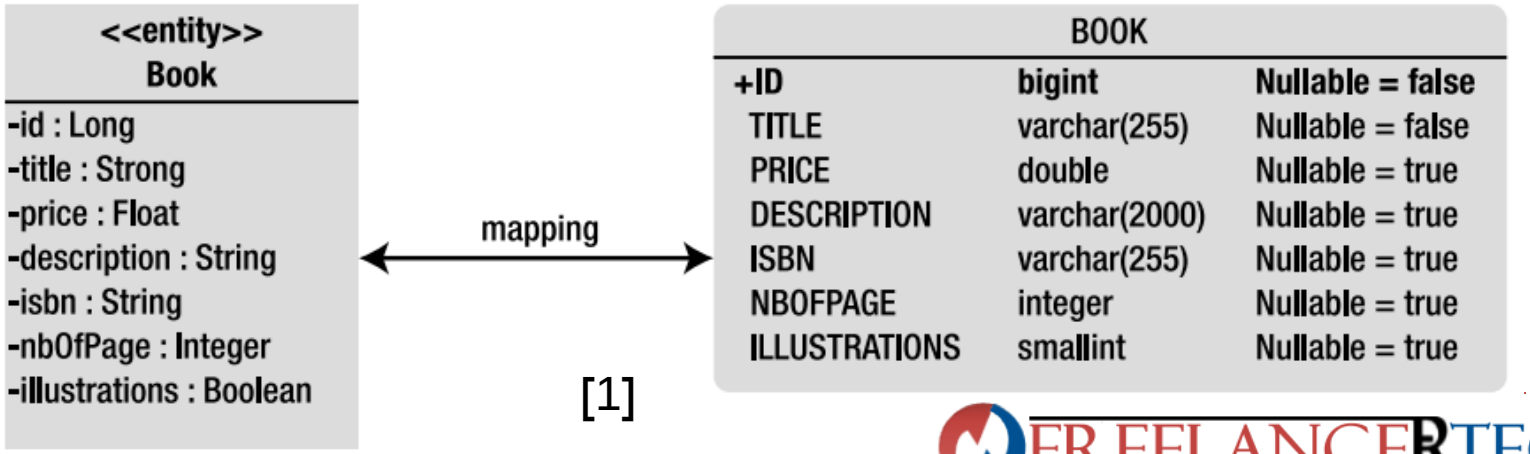
Ses principaux avantages sont les suivants :

- JPA peut être utilisé par toutes les applications Java, Java SE ou Java EE.
- Mapping O/R (objet-relationnel) avec les tables de la BD, facilitée par les Annotations.
- Un langage de requête objet standard JPQL pour la récupération des objets,

select ***p*** *from* ***Personne p*** *order by* ***p.nom*** *asc*

Entity: Entités

```
@Entity
@Table(name = "BOOK")
public class Book {
    @Id
    @GeneratedValue
    private Long id;
    @Column(name = "TITLE",nullable = false)
    private String title;
    private Float price;
    @Basic(fetch = FetchType.LAZY)
    @Column(length = 2000)
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations

   //Les Getters et les Setters
```



[1]

FREELANCERTECH

Autres annotations

- Voir par exemple : JPA Reference
  http://www.objectdb.com/api/java/jpa

- Les curieux peuvent consulter la spécification java EE 6 ou le tutorial (ou un bon livre)

Contexte de persistance

Ensemble des instances d'entités gérées a un instant donné,

gérées par qui ? Le gestionnaire d'entités : EntityManager

Ce contexte peut donc être considéré comme **un cache de premier niveau** : c'est un espace réduit ou le gestionnaire stocke les entités avant d'écrire son contenu dans la base de données,

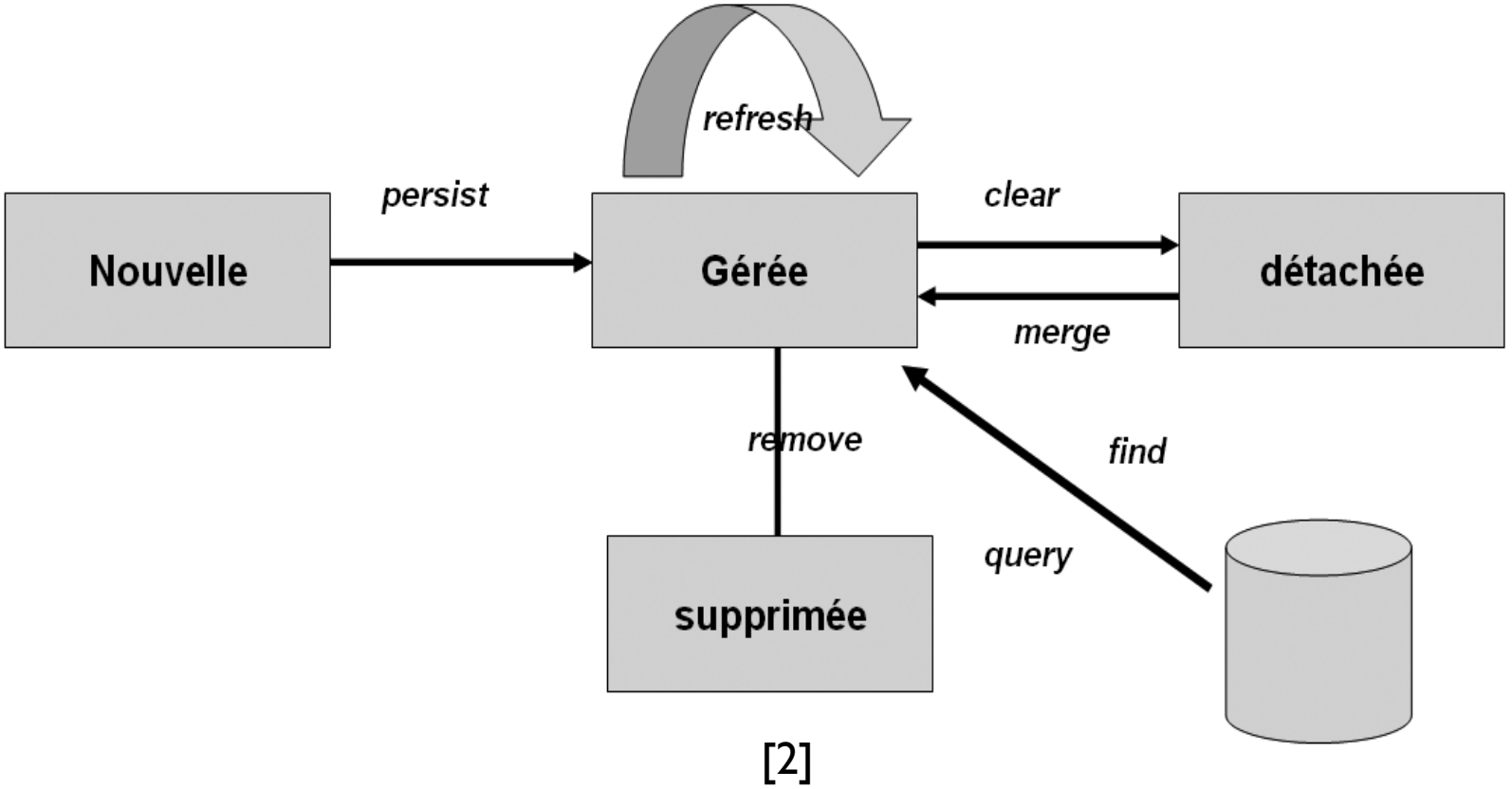
Opérations prises en charge par le gestionnaire d'entités

Opération	Description
persist()	Insère l'état d'une entité dans la base de données. Cette nouvelle entité devient alors une entité gérée.
remove()	Supprime l'état de l'entité gérée et ses données correspondantes de la base.
refresh()	Synchronise l'état de l'entité à partir de la base, les données de la BD étant copiées dans l'entité.
merge()	Synchronise les états des entités « détachées » avec le PC. La méthode retourne une entité gérée qui a la même identité dans la base que l'entité passée en paramètre, bien que ce ne soit pas le même objet.
find()	Exécute une requête simple de recherche de clé.
CreateQuery()	Crée une instance de requête en utilisant le langage JPQL.
createNamedQuery()	Crée une instance de requête spécifique.
createNativeQuery()	Crée une instance de requête SQL.
contains()	Spécifie si l'entité est managée par le PC.
flush()	Toutes les modifications effectuées sur les entités du contexte de persistance gérées par le gestionnaire d'entités sont enregistrées dans la BD lors d'un flush du gestionnaire.

[2]

NB : Un flush() est automatiquement effectué au moins à chaque **commit de la transaction** en cours,

Cycle de vie d'une instance d'entité



[2]

Obtention d'une fabrique EntityManagerFactory

L' interface EntityManagerFactory permet d'obtenir une instance de l'objet EntityManager,

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");
```

« *jpa* » est le nom de l'unité de persistance, définie dans le fichier de configuration de la couche JPA *METAINF*/persistence.xml.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence">
3.   <persistence-unit name="jpa" transaction-type="RESOURCE_LOCAL">
4.      <!--  provider -->
5.      <provider>org.hibernate.ejb.HibernatePersistence</provider>
6.      <properties>
7.         <!-- Classes persistantes -->
8.         <property name="hibernate.archive.autodetection" value="class, hbm" />
9.         <!-- logs SQL
10.           <property name="hibernate.show_sql" value="true"/>
11.           <property name="hibernate.format_sql" value="true"/>
12.           <property name="use_sql_comments" value="true"/>
13.        -->
14.        <!-- connexion JDBC -->
15.        <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver" />
16.        <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/jpa" />
17.        <property name="hibernate.connection.username" value="jpa" />
18.        <property name="hibernate.connection.password" value="jpa" />
19.        <!--  création automatique du schéma -->
20.        <property name="hibernate.hbm2ddl.auto" value="create" />
21.        <!-- Dialecte -->
22.        <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect" />
23.        <!--  propriétés DataSource c3p0 -->
24.        <property name="hibernate.c3p0.min_size" value="5" />
25.        <property name="hibernate.c3p0.max_size" value="20" />
26.        <property name="hibernate.c3p0.timeout" value="300" />
27.        <property name="hibernate.c3p0.max_statements" value="50" />
28.        <property name="hibernate.c3p0.idle_test_period" value="3000" />
29.     </properties>
30.  </persistence-unit>
31. </persistence>
```


FREELANCERTECH

Création du gestionnaire d'entité EntityManager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");

EntityManager  em  =  emf.createEntityManager() ;
```

Exemple d'insertion d'un livre

Version Java SE

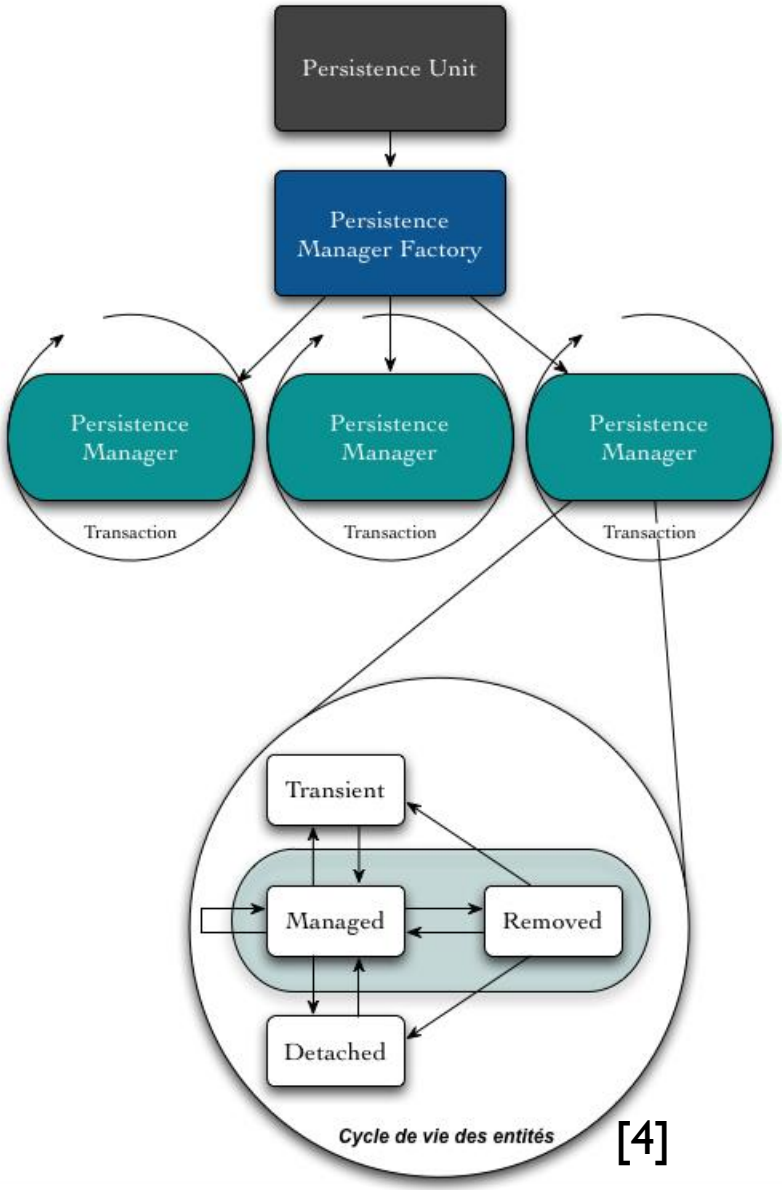
```
public class Main {

 public static void main(String[] args) {

  // On crée une instance de livre
  Book book = new Book();
  book.setTitle("MUGC: JPA\MYSQL");
  book.setPrice(12.5F);
  book.setDescription("Science fiction");
        …
 // On récupère un pointeur sur l'entity manager
 // Remarque : dans une appli web, pas besoin de
    faire tout cela !
 EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("jpa");
 EntityManager em = emf.createEntityManager();

 // On rend l'objet « persistant » dans la base (on
    l'insère)
 EntityTransaction tx = em.getTransaction();
 tx.begin();
  em.persist(book);
 tx.commit();

 em.close();
 emf.close();

 }
```

Version Java EE

```
@Stateless
public class BookBean {

    @PersistenceContext(unitName = "jpa")
    private EntityManager em;

    public void createBook() {
          Book book = new Book();
        book.setTitle("MUGC: JPA\MYSQL");
        book.setPrice(12.5F);
        book.setDescription("Science fiction");
        book.setIsbn("1-84023-742-2");
        book.setNbOfPage(354);
        book.setIllustrations(false);

        em.persist(book);

        // Récupère le livre dans la BD par sa clé
    primaire
        book = em.find(Book.class, 1234L);

        System.out.println(book);
    }
}
```

[1]

Fonctionnement de JPA



[4]

• La **Persistence Unit** : organise les meta données qui définissent le mapping entre les entités et la base de donnée dans le fichier de configuration *METAINF*/persistence.xml

• La **Persistence Manager Factory** récupère ces metas données de la Persistence Unit et les interprètent pour créer des Persistence Manager (EntityManager)

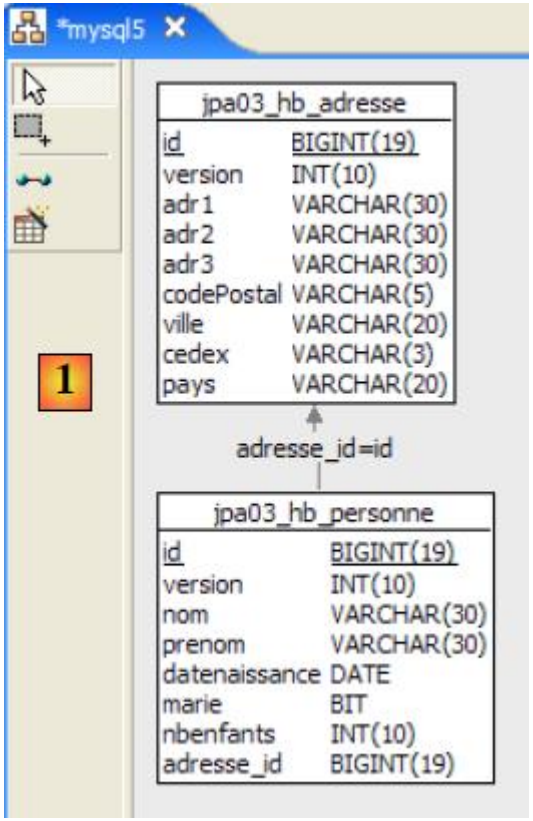
• Le **Persistence Mangager** (EntiyManager) gère les échanges entre le code et la base de donnée, c'est à dire le cycle de vie des **entités**

• Enfin, les opérations du EntityManager est englobé dans une **Transaction.**

FREELANCERTECH

Relations entre entités

Relation un-à-un



[3]

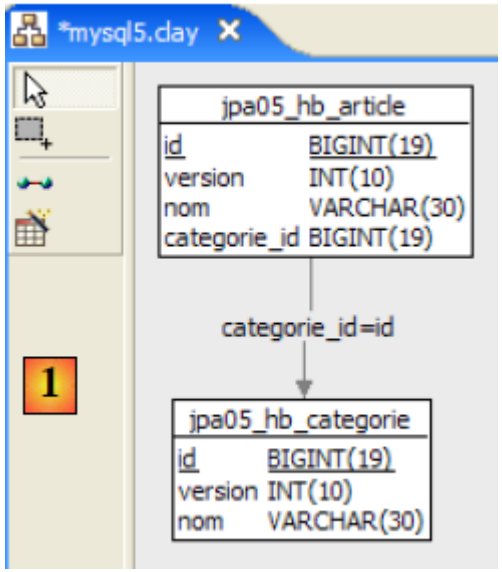
```
@Entity
@Table(name = "jpa03_hb_personne")
 public class Personne {
      ...

     @OneToOne(cascade = CascadeType.ALL, fetch=FetchType.LAZY)
      /*@OneToOne(cascade = {CascadeType.MERGE,        CascadeType.PERSIST,
       CascadeType.REFRESH, CascadeType.REMOVE}, fetch=FetchType.LAZY) */
     @JoinColumn(name = "adresse_id", unique = true, nullable = false)
      private Adresse adresse;
      ...

}
```

```
@Entity
@Table(name = "jpa03_hb_adresse")
public class Adresse{
     ...
     /*n'est pas obligatoire*/
    @OneToOne(mappedBy = "adresse", fetch=FetchType.EAGER)
     private Personne personne;
      ...

}
```


FREELANCERTECH

Relations entre entités

Relation un-à-plusieurs et plusieurs-à-un



[3]

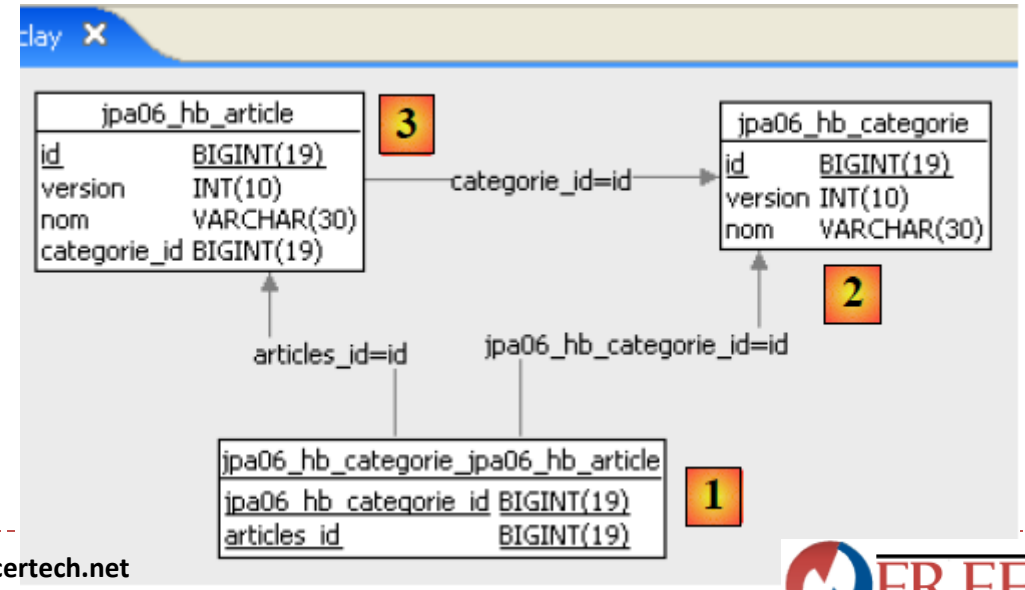
```
@Entity
@Table(name="jpa05_hb_article")
 public class Article{
      ...
    // relation principale Article (many) -> Category (one) implémentée par une clé
    // étrangère (categorie_id) dans Article
    // 1 Article a nécessairement 1 Categorie (nullable=false)

    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name = "categorie_id", nullable = false)
     private Categorie categorie;
     ...
}
```

```
@Entity
@Table(name="jpa05_hb_categorie")
public class Categorie{
    ...
   // relation inverse Categorie (one) -> Article (many) de la relation Article (many)– > Categorie (one)
   @OneToMany(mappedBy = "categorie", cascade = { CascadeType.ALL })
    private Set<Article> articles = new HashSet<Article>();
    ...

}
```

Relations entre entités

Relation un-à-plusieurs et plusieurs-à-un

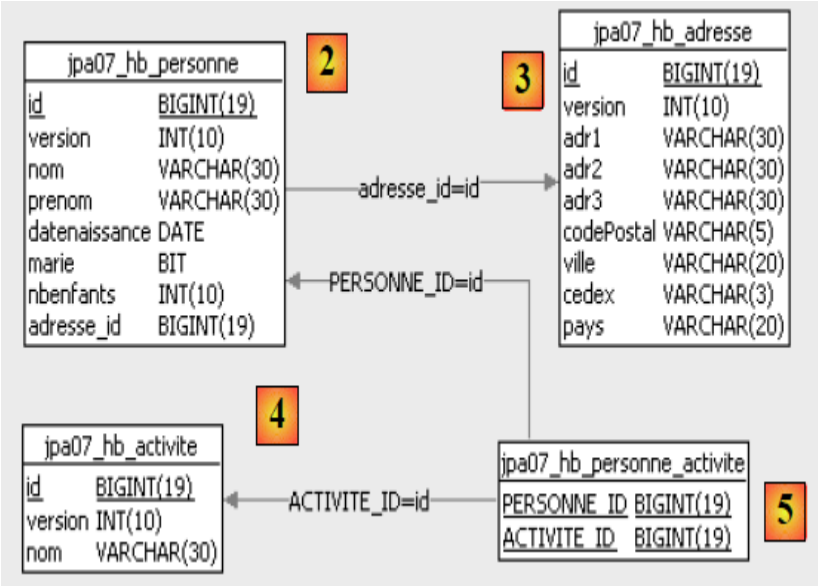
```
@Entity
@Table(name="jpa06_hb_categorie")
public class Categorie{
    ...
    // relation OneToMany non inverse (absence de mappedby) Categorie (one) -> Article (many)
    // implémentée par une table de jointure Categorie_Article pour qu'à partir  d'une catégorie
    // on puisse atteindre plusieurs articles

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private Set<Article> articles = new HashSet<Article>();
    ...

}
```



[3]

Relations entre entités

Relation plusieurs-à-plusieurs



pratiquer [3]

```
@Entity
@Table(name = "jpa07_hb_personne")
 public class Personne {
      ...
    // relation Personne (many) -> Activite (many) via une table de jointure
      personne_activite
    @ManyToMany(cascade={CascadeType.PERSIST})
    @JoinTable(name="jpa07_hb_personne_activite",
                    joinColumns = @JoinColumn(name = "PERSONNE_ID"),
                    inverseJoinColumns = @JoinColumn(name =
"ACTIVITE_ID"))

      private Set<Activite> activites = new HashSet<Activite>();
       ...

}
```

```
@Entity
@Table(name = "jpa07_hb_activite")
public class Activite{
    ...

    // relation inverse Activite -> Personne
    @ManyToMany(mappedBy = "activites")
    private Set<Personne> personnes = new HashSet<Personne>();
...

}
```

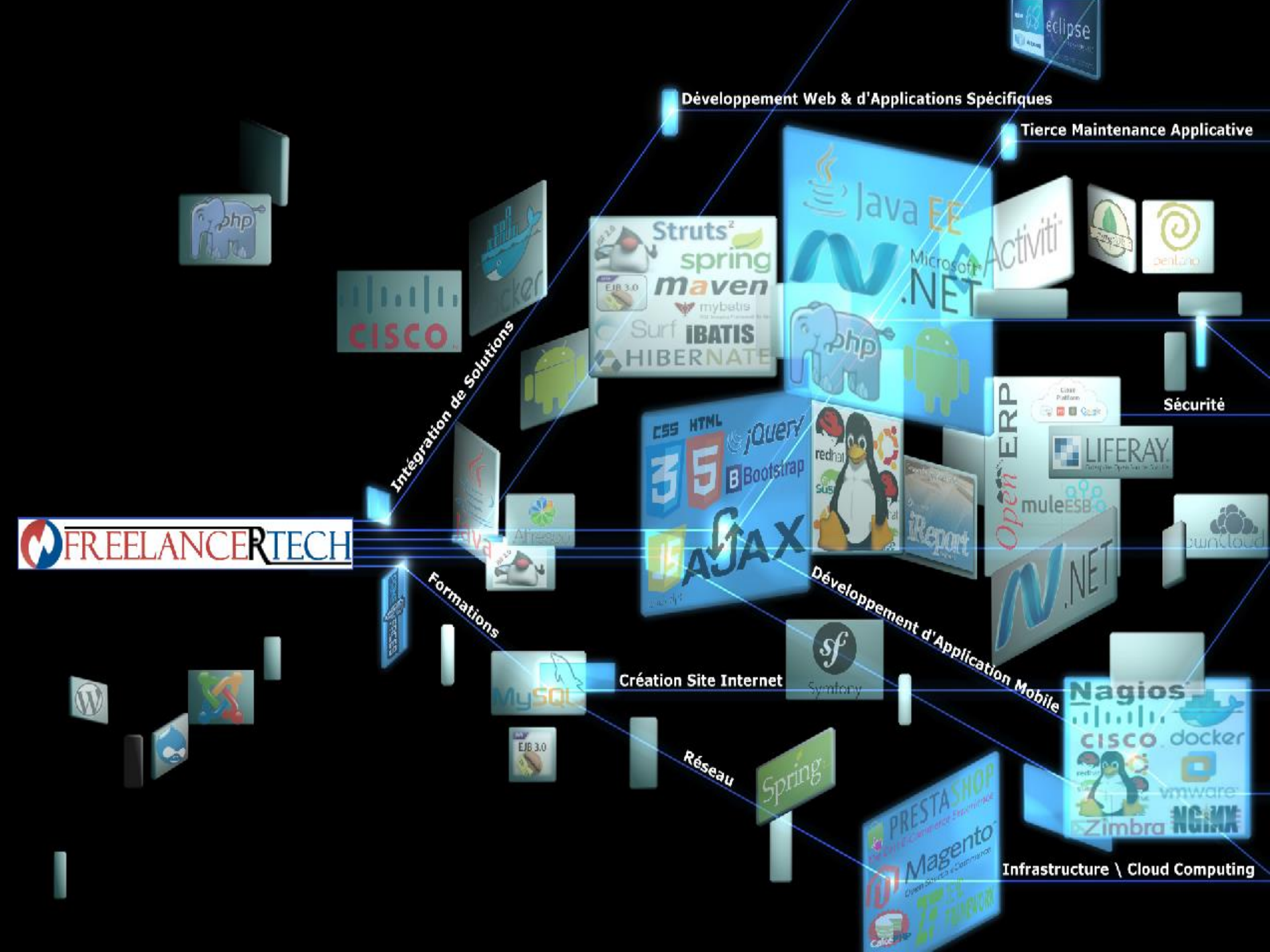
Travaux pratiques

Génération des entités à partie d'une base de donnée MySQL avec NetBean

Références

- [1] www.docstoc.com/docs/150704719/5--JPA-Michel-2012ppt
- [2] www.eyrolles.com/Chapitres/9782212120615/Chap10_Djaafar.pdf
- [3] www.tahe.ftp-developpez.com/fichiers-archive/jpa.pdf
- [4] www.blog.ippon.fr/2011/10/11/jpa-une-magie-qui-se-merite-retour-aux-sources-de-jpa/

Questions


Développement Web & d'Applications Spécifiques
Tierce Maintenance Applicative
Intégration de Solutions
Sécurité
FREELANCERTECH
Formations
Développement d'Application Mobile
Création Site Internet
Réseau
Infrastructure \ Cloud Computing