Bean Entité

Gestion de la persistance

Plan

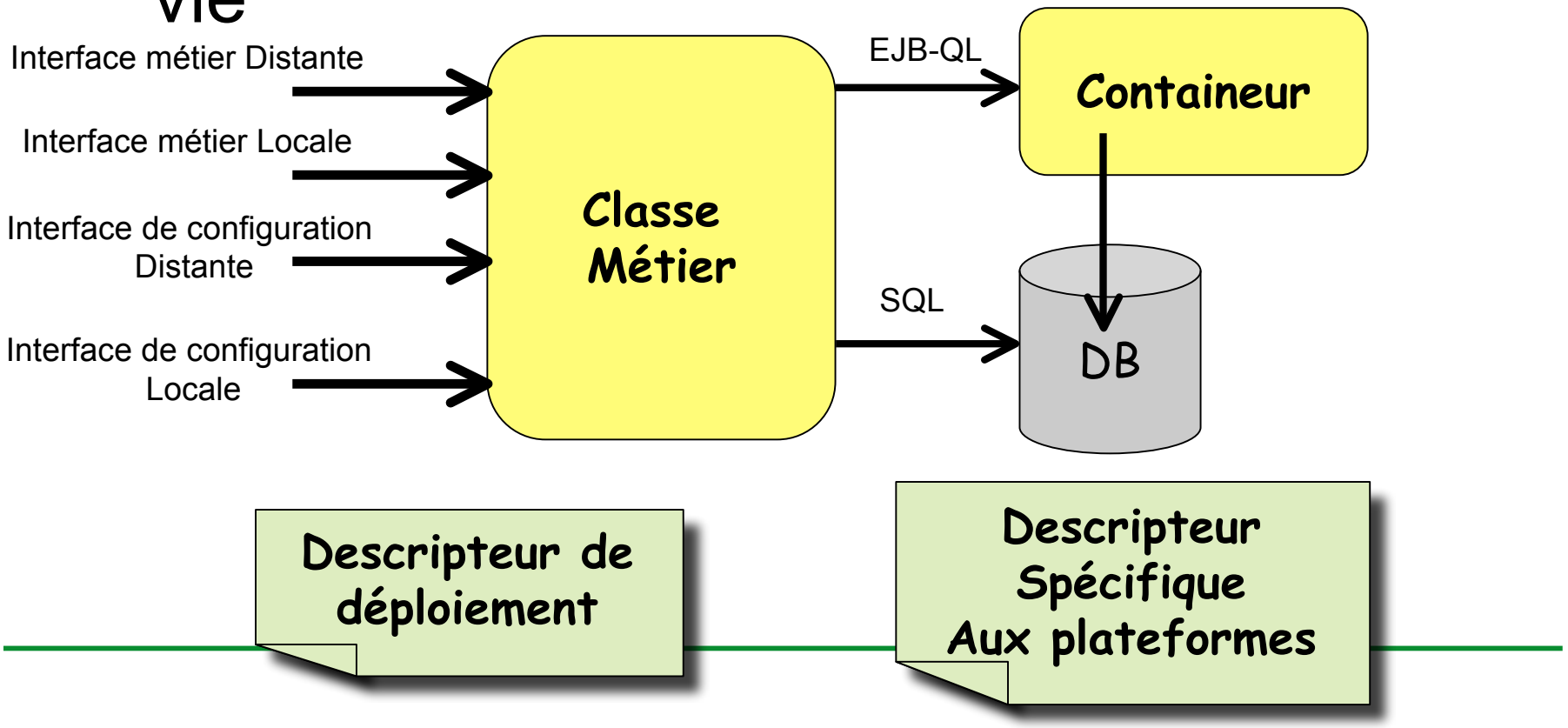
- Généralités sur le bean entité
- Gestion de la persistance en EJB3
- Développer une entité persistante
- Manipuler le gestionnaire de persistance
- Les requêtes & JPA-QL
- Relations entre entités

Bean entité

- Objectifs
  - Permettre à plusieurs utilisateurs d'accéder aux mêmes données
  - Stocker et/ou accéder à des données
- Particularité
  - Lié à une base de données

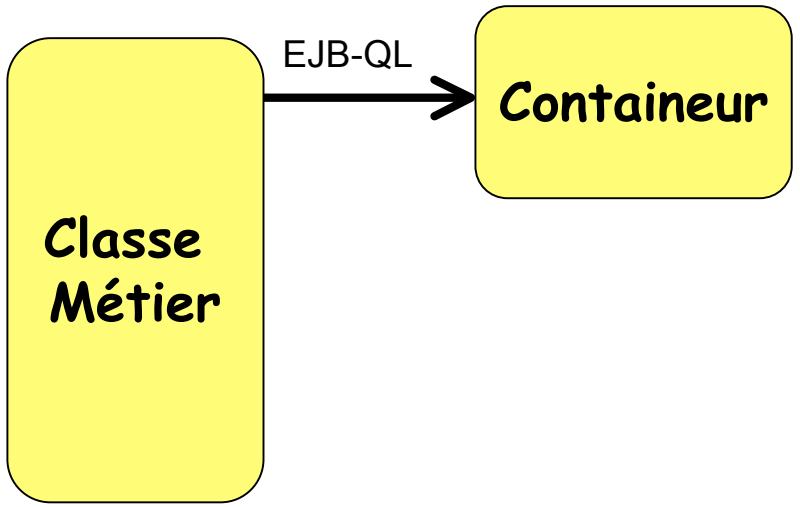
Bean entité : EJB 2

- Un bean à part entière avec un cycle de vie



Bean entité : EJB 3

- Dans la norme EJB 3, les entités sont des POJO
  - Plain Old Java Objects
- Des objets
  - classiques
  - sérialisés
  - annotés



- Des représentations JAVA d'une ou plusieurs table(s)

Bean Entité : EJB3 vs EJB2

- En EJB 3
  - Pas d'interfaces métiers
  - Pas d'interfaces de configuration
  - Les entités sont accessibles uniquement via le gestionnaire de persistance
- Gestion de la persistance externe
  - Entity Manager
    - Nouveauté des EJB 3

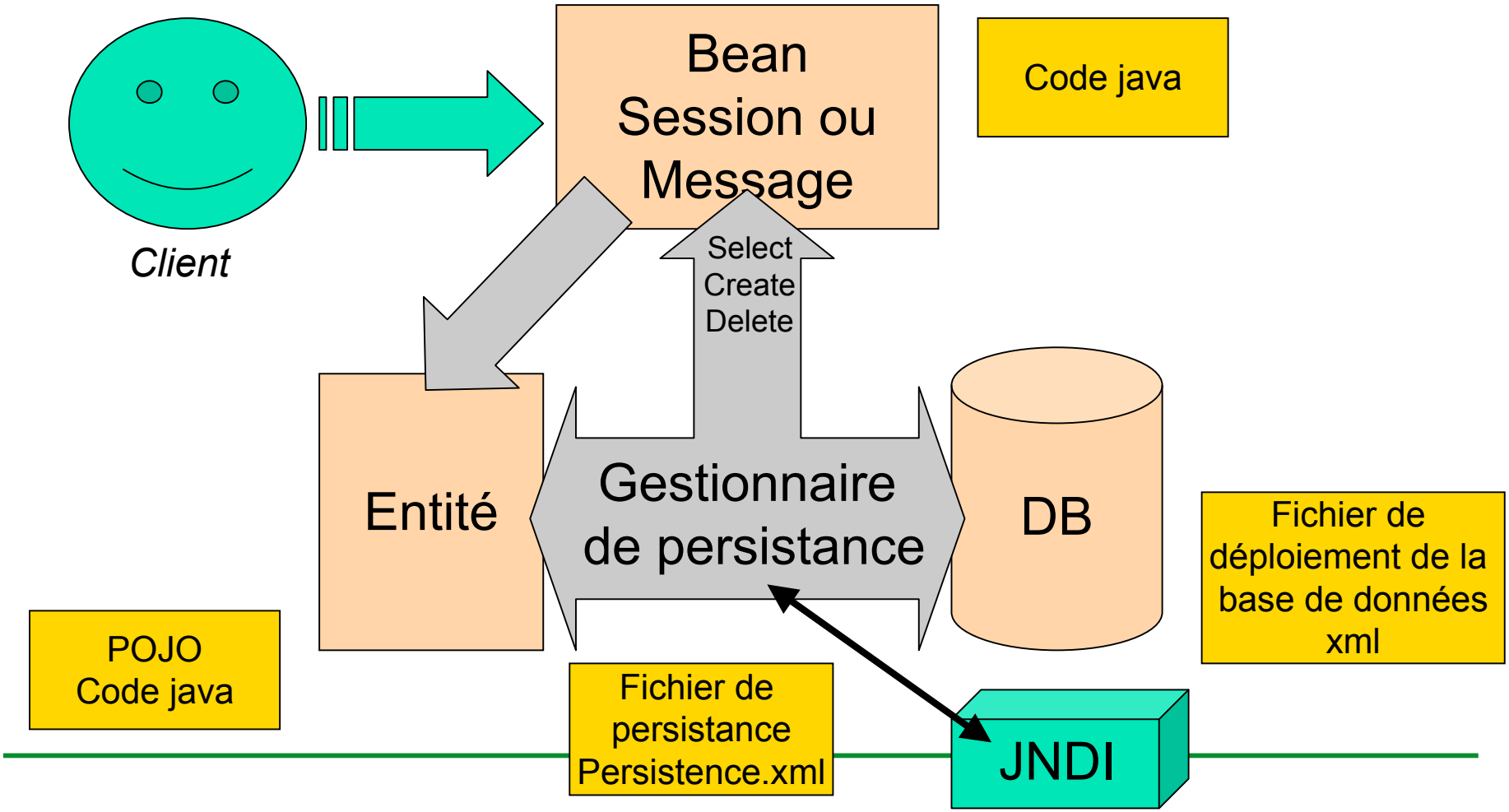
Gestion de la persistance en EJB3

Exemple d’un annuaire de personnes

Le gestionnaire de persistance

- Pas d'accès direct aux entités
  - L'accès aux entités s'effectue au travers du gestionnaire de persistance
- JPA (Java Persistence API)
  - Permet le mapping entre
    - Les objets java J2EE
    - Base de données relationnelles
  - Evolution de Hibernate
    - Ancien gestionnaire de persistance
- Avantage
  - Souplesse d'écriture du code de persistance
  - Utilisation possible de l'héritage des objets
  - Possibilité de déploiement dans ou hors un conteneur EJB

Gestion de persistance EJB 3



Base de données

- Structure de la table « Personne »

Attribut	Type
codePersonne	Integer
nom	Varchar(60)
prenom	Varchar(60)

L'entité Personne

```
package database.personnes;

@Entity
public class Personne implements Serializable
{
      @Id
      private int codePersonne;
      private String nom;  …

      public Personne() { super(); }

      public int getCodePersonne() { return this.c
      public void setCodePersonne(int codePersonne) { this.codePersonne = codePersonne;}
      public String getNom() { return this.nom; }
      public void setNom(String nom) { this.nom = nom; } …
 }
```

Par défaut la table liée porte le nom de la classe

Clé primaire

Constructeur par défaut obligatoire

Classe Personne

Quelques annotations importantes

Annotation	Description
@Entity	La classe est un bean entité
@Id	L'objet ou la fonction qui suit est la clé primaire de l'entité
@generatedValue[strategy="…"]	Définit une politique de génération automatique de la clé-primaire
@Table[name="…"]	Assigne l'entité à une table spécifique
@Column[name= "…"]	Assigne l'attribut qui suit à une colonne spécifique de la table

L’entité Personne v2

```
package database.personnes;

@Entity
@Table(name=«Personne»)
public class  Personne implements Serializable
{
      @Id @generatedValue
      @column(name=«code_personne »)
      private int codePersonne;
      @Column(name=«nom» , length=60)
      private String nom;
      public  Personne() { super(); }
      public  int getCodePersonne() { return this.codePersonne;      }
      public  void setCodePersonne(int codePersonne) { this.codePersonne = codePersonne;}
      public  String getNom() { return this.nom; }
      public  void setNom(String nom) { this.nom = nom; }
 }
```

Classe Personne

Déclaration de l'entité dans le gestionnaire de persistance

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
   xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xm
   http://java.sun.com/xml/ns/persistence/per
<persistence-unit name="TP_PAD">
        <jta-data-source>java:jndi/TP_PAD</jta-data-source>

        <class>database.personnes.Personne</class>
</persistence-unit>
</persistence>
```

Nom JNDI
De la source de
données

Nom de la classe

Fichier persistence.xml

Déploiement de la source de données

```
<datasources>
  <local-tx-datasource>
    <jndi-name>jndi/TP_PAD</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/TP_PAD</connection-url>
    <driver-class>org.gjt.mm.mysql.Driver</driver-class>
    <user-name>root</user-name>
    <password></password>
    <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-
       sorter-class-name>
    <!-- sql to call when connection is created
    <new-connection-sql>some arbitrary sql</new-connection-sql>
      -->
    <!-- sql to call on an existing pooled connection when it is obtained from pool
    <check-valid-connection-sql>some arbitrary sql</check-valid-connection-sql>
      -->

    <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml (optional) -->
    <metadata>
      <type-mapping>mySQL</type-mapping>
    </metadata>
  </local-tx-datasource>
</datasources>
```

Mysql-ds.xml

Le fichier Mysql-ds.xml doit être copié dans le dossier « deploy » de JBOSS

Manipuler les beans entités

Manipuler des données dans une base de données

Manipulation du gestionnaire de persistance

- Accès direct à l'EM (entityManager)
  - Par injection

```
@PersistenceContext(unitname=« … »)
private EntityManager em;
```

- Fonction simple de l'EM
  - Persister un objet (créer une nouvelle entrée dans la base)
    - Em.persist(objEntity)
  - Supprimer un objet
    - Em.remove(objEntity)
  - Rechercher un objet
    - Em.find(objEntity.class, cle) //retourne null si non trouvé
  - Forcer la mise à jour de la base de données
    - Em.flush()

Exemple

- Création d'un bean session
  - Catégorie
    - Stateless
  - Objectif
    - Manipuler l'entité « Personne »
- Interface

```
package managers;
import javax.ejb.Remote;
@Remote
public interface PersonneManager
{
    public void creerPersonne(String nom, String prenom);
    public Personne rechercherPersonne(int clef);
    public void supprimerPersonne(int clef);
    public void setNomPersonne(int clef, String nom);
    public void setPrenomPersonne(int clef, String nom);
}
```

Exemple : fonction d'ajout

```
package managers;
import javax.ejb.Stateless;
@Stateless
public class PersonnesManagerBean implements PersonnesManager
{
    @PersistenceContext
    private EntityManager em;

    public void creerPersonne(String nom, String prenom)
    {
          Personne p= new Personne();
          p.setNom(nom);
          p.setPrenom(prenom);
          this.em.persist(p);
    }
}
```

Exemple : fonctions de recherche et de suppression

```
public class PersonnesManager implements PersonnesManagerRemote
{
    @PersistenceContext
    private EntityManager em;
          (…)
    public void rechercherPersonne(int clef)
    {
          Personne p= em.find(Personne.class, clef);
          return p;
    }
    public void SupprimerPersonne(int clef)
    {
          Personne p= em.find(Personne.class, clef);
          em.remove(p);
    }
}
```

Exemple : fonctions de mise à jour

```
public class PersonnesManager implements PersonnesManagerRemote
{
    @PersistenceContext
    private EntityManager em;
          (…)
    public void setNomPersonne(int clef, String nom)
    {
          Personne p= em.find(Personne.class, clef);
          p.setNom(nom);
          em.flush();
    }
    public void setPrenomPersonne(int clef, String prenom)
    {
          Personne p= em.find(Personne.class, clef);
          p.setPrenom(prenom);
    }
}
```

ATTENTION !
Une mise à jour n'implique pas une modification « physique » de la base de données

Requêtes & JPA-QL

Requête & JPA-QL

- Objectifs
  - Effectuer des recherches spécifiques (paramétrées) de données persistantes
- Interrogation du gestionnaire de persistance
  - Requêtes simples
    - Requête ad-hoc (définit dans le code de « l'appelant »)
  - Requêtes nommée
    - Requête prédéfinie (définit dans le code de l'entité)
- JPA-QL = EJB-QL
  - SQL-like
    - Langage spécifique au domaine des EJB
  - Langage orienté objet
    - Le résultat d'une requête EJB-QL est un objet ou une liste d'objet
    - Le recours aux jointures est limité

JPA-QL en bref (1/3)

- Syntaxe
  - complète
    - SELECT OBJECT(p) From Personne AS p
  - simplifié
    - From Personne p
- Extraction d'attribut d'entités
  - SELECT p.nom From Personne AS p
- Recomposition d'objet
  - SELECT new NomPers(p.nom,p.prenom) From Personne AS p

JPA-QL en bref (2/3)

- Clause FROM Similaire à SQL
  - Inner Join, Left Join, …
- Clause WHERE similaire SQL
  - LIKE, BETWEEN, =, AND, OR ...
  - [NOT] IN (liste valeurs) , IS [NOT] NULL
  - IS [NOT] EMPTY, [NOT] MEMBER OF

JPA-QL en bref (3/3)

- Aggrégation
  - Count, max, min, avg, sum, having, ….
- Expressions fonctionnelles
  - LOWER(Sttring), UPPER(String),
  - LENGTH(String), LOCATE(String1,String2 [,start])
  - CONCAT(String1,String2),

  SUBSTRING(String,start,length)
  - ABS(number) SQRT(double)
  - CURRENT_DATE CURRENT_TIME CURRENT_TIMESTAMP

Requête simple (1/2)

- Requête définit de manière ad-hoc dans le code de l'appelant
  - Un bean session
  - Un client …

- Une instance spécifique

```
Query q=em.createQuery("from Personne c where codePersonne
    =110");
 Personne c=(Personne )q.getSingleResult();
```

- Une collection

```
Query q=em.createQuery("from Categorie c");
List<Categorie> l=q.getResultList();
```

Requête simple (2/2)

- 2 modes de paramétrage
  - Paramètre nommé
    - Query q=em.createQuery(“from Personne e where e.codePersonne=:id”);
    - q.setParameter(“id”,123);
  - Paramètre numéroté
    - Query q=em.createQuery(“from Personne e where e.codePersonne=?1”);
    - q.setParameter(1,123);

API de Query

```
package javax.persistence;

public interface Query {
    public List getResultList( );          //fonctions de
    public Object getSingleResult( );  //sélection d'entités
    public int executeUpdate( );   //Fonction de mise à jour de la base (delete, insert, update)
    public Query setMaxResults(int maxResult);
    public Query setFirstResult(int startPosition);
    public Query setHint(String hintName, Object value);
    public Query setParameter(String name, Object value);
    public Query setParameter(String name, Date value, TemporalType temporalType);
    public Query setParameter(String name, Calendar value, TemporalType temporalType);
    public Query setParameter(int position, Object value);
    public Query setParameter(int position, Date value, TemporalType temporalType);
    public Query setParameter(int position, Calendar value, TemporalType temporalType);
    public Query setFlushMode(FlushModeType flushMode);
    }
```

Exemple

```
public class PersonnesManagerBean implements PersonnesManager{
    @PersistenceContext
    private EntityManager em;
          (…)
    public Personne rechercherPersonne(int clef) {
          Personne p= em.find(Personne.class, clef);
          return p;
    }
    public List<Personne> rechercherPersonneNom(String nom) {
          Query q= em.createQuery("FROM Personne p WHERE p.nom like :pNom");
          q.setParameter("pNom",nom).setMaxResults(10);
          return q.getResultList();
    }
}
```

Requêtes nommées

- Requêtes prédéfinies
  - Implémentées dans
    - le code de l'entité via les annotations
    - le descripteur de déploiement (ejb-jar.xml)

- Annotations utiles

```
@NamedQueries({
    @NamedQuery(name= « … », query= « … »)
    @NamedQuery(name= « … », query= « … »)
})
```

Exemple (1/2)

```
package database.personnes;

@NamedQueries({
    @NamedQuery(name=« rechercheParNom », query=« From Personne p Where p.nom=:pnom)
})
@Entity
@Table(name=«Personne»)
public class PersonneBean implements Serializable
{
      @Id @generatedValue
      @column(name=«code_personne »)
      private int codePersonne;
      @Column(name=«nom» , length=60)
      private String nom;
      public Personne() { super(); }
      public int getCodePersonne() { return this.codePersonne;  }
      public void setCodePersonne(int codePersonne) { this.codePersonne = codePersonne;}
      public String getNom() { return this.nom; }
      public void setNom(String nom) { this.nom = nom; }
}
```

Code métier de l'entité

Exemple (2/2)

```
public class PersonnesManagerBean implements PersonnesManager{
    @PersistenceContext
    private EntityManager em;
          (…)
    public Personne rechercherPersonne(int clef) {
          Personne p= em.find(Personne.class, clef);
          return p;
    }
    public void rechercherPersonneNom(String nom) {
          Query q= em.createNamedQuery(« rechercheParNom »);
          q.setParameter(“pNom”,nom);
          q.setMaxResults(10);
          q.getResultList();
    }
}
```

Code métier du bean session

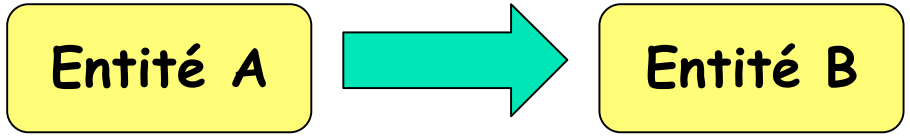
Relation entre entités

Relation entre entités

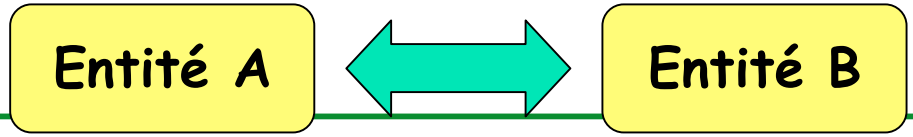
- Objectifs
  - Associer simplement une entité à une autre entité
  - Reproduire les relations entre des tables d'une base de données
    - 1-1, 1-n, n-n
    - Mises à jour en cascade

Relations monodirectionnelle vs bidirectionnelle

- Relation monodirectionnelle
  - La relation ne peut être parcourue que dans un sens



- Relation bidirectionnelle
  - La relation peut être parcourue dans les deux sens



Taxonomie des relations

- One-to-one monodirectionnelle
- One-to-one bidirectionnelle
- One-to-many monodirectionnelle
- Many-to-one monodirectionnelle
- Many-to-many monodirectionnelle
- Many-to-many bidirectionnelle

Exemple : personne/adresse

- Schéma de la base de données
  - Une personne est associés à une adresse

Table Personne	
Attribut	Type
Code_personne	Integer
nom	Varchar(60)
prenom	Varchar(60)
adresse	Integer

Table Adresse	
Attribut	Type
adresseId	Integer
numero	Integer
rue	Varchar(255)
codePostal	Varchar(5)
ville	Varchar(60)

Relation One-to-One

- Objectif
  - Relation 1-1
  - Une entité est liée à une et une seule autre entité
- Liaison entre les tables
  - Une dépendance de référence est définie dans la table qui initie la relation
- Annotation
  - @OneToOne

Relation One-to-One

- Syntaxe

```
@OneToOne
   @JoinColumn(name= « col »)
private TypeEntité maVar;
```

  - « col » : nom de la clé étrangère dans la table

Relation One-to-One Monodirectionnelle

```java
@Entity
public class Personne
   implements Serializable {
    @Id
    @GeneratedValue
    @Column(name="code_personne")
    private int codePersonne;
    @Column(name="nom", length=60)
    private String nom;
    private String prenom;
    @OneToOne
    @JoinColumn(name="adresse »
      referencedColumne=« idAdresse
       »)
    private Adresse adresses;
    …
```

```java
@Entity
public class Adresse
   implements Serializable {
    @Id
    @GeneratedValue
    private int idadresse;
    private int numero;
    private String rue;
    private String ville;
    private String codepostale;
    …
```

Relation One-to-One Bidirectionnelle

```java
@Entity
public class Personne
   implements Serializable {
    @Id
    @GeneratedValue
    @Column(name="code_personne")
    private int codePersonne;
    @Column(name="nom", length=60)
    private String nom;
    private String prenom;
    @OneToOne
    @JoinColumn(name="adresse")
    private Adresse mAdresse;
    …
```

```java
@Entity
public class Adresse implements
   Serializable {
    @Id
    @GeneratedValue
    private int idadresse;
    private int numero;
    private String rue;
    private String ville;
    private String codepostale;
    @OneToOne(mappedBy="mAdresse")
    private Personne personne;
    …
```

Du côté de l'entité possédant la clé étrangère

Relation One-to-Many monodirectionnelle

- Objectif
  - Relation 1-n
  - Accéder à plusieurs entités à partir d'une seule entité
- Liaison entre les tables
  - Une dépendance de référence est définie dans la table « many »
- Annotation

```
@OneToMany
```

Relation One-to-Many monodirectionnelle

- Syntaxe :

```
@OneToMany
    @JoinColumn(name= « col »)
private Collection<TypeEntité> maVar = new
  Vector<TypeEntité>();
```

  - « col » : nom de la clé étrangère dans la table référencée

Exemple : table téléphone

- Schéma de la base de données
  - Une personne est associés à un ou plusieurs numéro de téléphone

Table Personne	
Attribut	Type
code_personne	Integer
nom	Varchar(60)
prenom	Varchar(60)
adresse	Integer

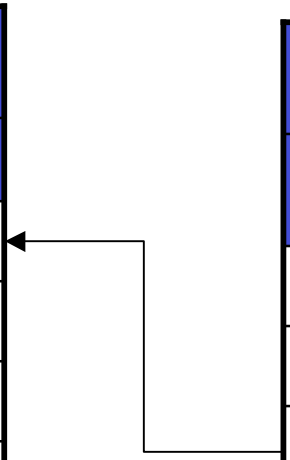


Table Téléphone	
Attribut	Type
telephoneID	Integer
numero	Varchar(10)
personneID	Integer

Relation One-to-Many monodirectionnelle

```java
@Entity
public class Personne implements
    Serializable {
    @Id
    @GeneratedValue
    @Column(name="code_personne")
    private int codePersonne;
    @Column(name="nom", length=60)
    private String nom;
    private String prenom;
    @OneToOne
    @JoinColumn(name="adresse")
    private Adresse adresses;
    @OneToMany
    @JoinColumn(name="personneID")
    private Collection<Telephone> telephones
       =new Vector<Telephone>();
    …
```

```java
@Entity
public class Telephone implements
    Serializable {
    @Id
    @GeneratedValue
    private int telephoneid;

    private String numero;

    @Column(name="personneID")
    private int personnesID;
    …
```

Colonne « personneID » de la table Telephone

Relation Many-to-One monodirectionnelle

- Objectif
  - Relation n-1
  - Accéder à une même entité à partir de plusieurs entités
- Liaison entre les tables
  - Une dépendance de référence est définie dans la table « many »
- Annotation

```
@ManyToOne
```

Relation Many-to-One monodirectionnelle : syntaxe

- Syntaxe

```
@ManyToOne
  @JoinColumn(name= « col » [,
  referencedColumnName=« col2 »])
private TypeEntité maVar;
```

  - « col » : nom de la clé étrangère dans la table de l'entité
  - « col2 » : nom de la clé dans la table référencée

Relation Many-to-One monodirectionnelle

```java
@Entity
public class Personne
   implements Serializable {
    @Id
    @GeneratedValue
    @Column(name="code_personne")
    private int codePersonne;
    @Column(name="nom", length=60)
    private String nom;
    private String prenom;
    @OneToOne
    @JoinColumn(name="adresse")
    private Adresse adresses;
    …
```

```java
@Entity
public class Telephone implements
   Serializable {
    @Id
    @GeneratedValue
    private int telephoneid;

    private String numero;

    @ManyToOne
       @JoinColumn(name="personneID")
    private Personnes personnes;
```

Relation One-to-Many/Many-to-One Bidirectionnelle

- Objectif
  - Relation 1-n / n-1
  - Une entité est à plusieurs autres entités
- Liaison entre les tables
  - Une dépendance de référence est définie dans la table « many »
- Annotations utiles

  `@ManyToOne`

  `@OneToMany`

Relation One-to-Many/Many-to-One Bidirectionnelle : syntaxe

- Entité 1 :

```
@ManyToOne
   @JoinColumn(name= « col »)
   private TypeEntité1 maVar ;
```

« col » : nom de la clé étrangère dans la table de l'entité

- Entité 2

```
@OneToMany (mappedName=« maVar »)
   private Collection<TypeEntité2> maVar2 = new
   Vector< TypeEntité2 >();
```

« maVar » : nom de la variable dans la première entité

Relation One-to-Many/Many-to-One Bidirectionnelle

```java
@Entity
public class Personne implements
    Serializable {
    @Id
    @GeneratedValue
    @Column(name="code_personne")
    private int codePersonne;
    @Column(name="nom", length=60)
    private String nom;
    private String prenom;
    @OneToOne
    @JoinColumn(name="adresse")
    private Adresse adresses;
    @OneToMany(mappedBy="personne")
    private Collection<Telephone> telephones
        =new Vector<Telephone>();
    …
```

```java
@Entity
public class Telephone implements
    Serializable {
    @Id
    @GeneratedValue
    private int telephoneid;

    private String numero;

    @ManyToOne
    @JoinColumn(name="personneID")
    private Personnes personne;
```

Relation Many-to-Many

- Objectif
  - Relation n-n
  - Accéder à plusieurs entités à partir de plusieurs entités

- Annotations

  `@ManyToMany`

Relation Many-To-Many : syntaxe

- Entité 1

```
@ManyToMany
@JoinTable(name=« TableName », joinColumns=
    @JoinColumn(name=« col1 »,
      referencedColumnName=« col2 »),
    inverseJoinColumns=
    @JoinColumn(name=« col3 »,
      referencedColumnName=« col4 »))
private Adresse maVar new Vector<Adresse>();
```

  - « tableName » : nom de la table de jointure
  - « col1 » : dépendance de référence vers entité 1
  - « col2 » : clé primaire de l'entité 1
  - « col3 » : dépendance de référence vers entité 2
  - « col4 » : clé primaire de l'entité 2

Relation Many-To-Many : syntaxe

- Entité 2

```
@ManyToMany(mappedBy=« maVar »)
private Collection<Personne> maVar2=new
   Vector<Personne>();
```

  - « maVar » : nom de la variable dans l'entité 1

Exemple :

Table Personne	
Attribut	Type
Code_personne	Integer
nom	Varchar(60)
prenom	Varchar(60)

Table Adresse	
Attribut	Type
adresseId	Integer
numero	Integer
rue	Varchar(255)
codePostal	Varchar(5)
ville	Varchar(60)

Table PersonneAdresse	
Attribut	Type
personneID	Integer
adresseID	Integer

Relation Many-to-Many

```
@Entity
public class Personne implements
    Serializable {
    …
    @ManyToMany
    @JoinTable(name="PersonneAdresse",
       joinColumns=
         @JoinColumn(name="personneID",
           referencedColumnName="code_per
           sonne"),
         inverseJoinColumns=
         @JoinColumn(name="adresseID",
           referencedColumnName=« adresse
           id"))
    private Adresse adresses=
    new Vector<Adresse>();
    …
```

```
@Entity
public class Adresse implements
    Serializable {
     @Id
     @GeneratedValue
     private int idadresse;
     private int numero;
     private String rue;
     private String ville;
     private String codepostale;
     @ManyToMany(mappedBy=« adresses")
     private Collection<Personne>
        personnes=new Vector<Personne>();
     …
```

Mise à jour en cascade

- Permet la mise à jour d'une entité liée lors de l'ajout, modification, suppression, rechargement de l'entité principale

- Syntaxes
  - @TypeRelation(cascade= CascadeType1})
  - @TypeRelation(cascade= {CascadeType1, CascadeType2, …})
  - TypeRelation = @OneToOne, @ManyToOne, @OneToMany, ManyToMany

Mise à jour en cascade

- CascadeType.PERSIST
  - Enregistrement automatique de l'entité liée
- CascadeType.MERGE
  - Modification automatique de l'entité liée
- CascadeType.REMOVE
  - Suppression automatique de l'entité liée
- CascadeType.REFRESH
  - Mise à jour automatique de l'entité liée à partir de la base de données
- CascadeType.ALL
  - Cumule les quatres types de mise à jour

Mise à jour en cascade

```
@Entity
public class Personne implements Serializable {
    …
    @ManyToMany(cascade=CascadeType.ALL)
    @JoinTable(name="PersonneAdresse", joinColumns=
        @JoinColumn(name="personneID",
          referencedColumnName="code_personne"),
        inverseJoinColumns=
        @JoinColumn(name="adresseID",
          referencedColumnName="idadresse"))
    private Adresse adresses=
    new Vector<Adresse>();
    …
```

Exercice

- Concevoir et développer un forum qui permettra la gestion :
  - Des sujets
  - Des messages
  - Des utilisateurs

- Etablir le schéma de la base de données et les composants qui permettent sa gestion