La spécification JPA 2.1
Les associations

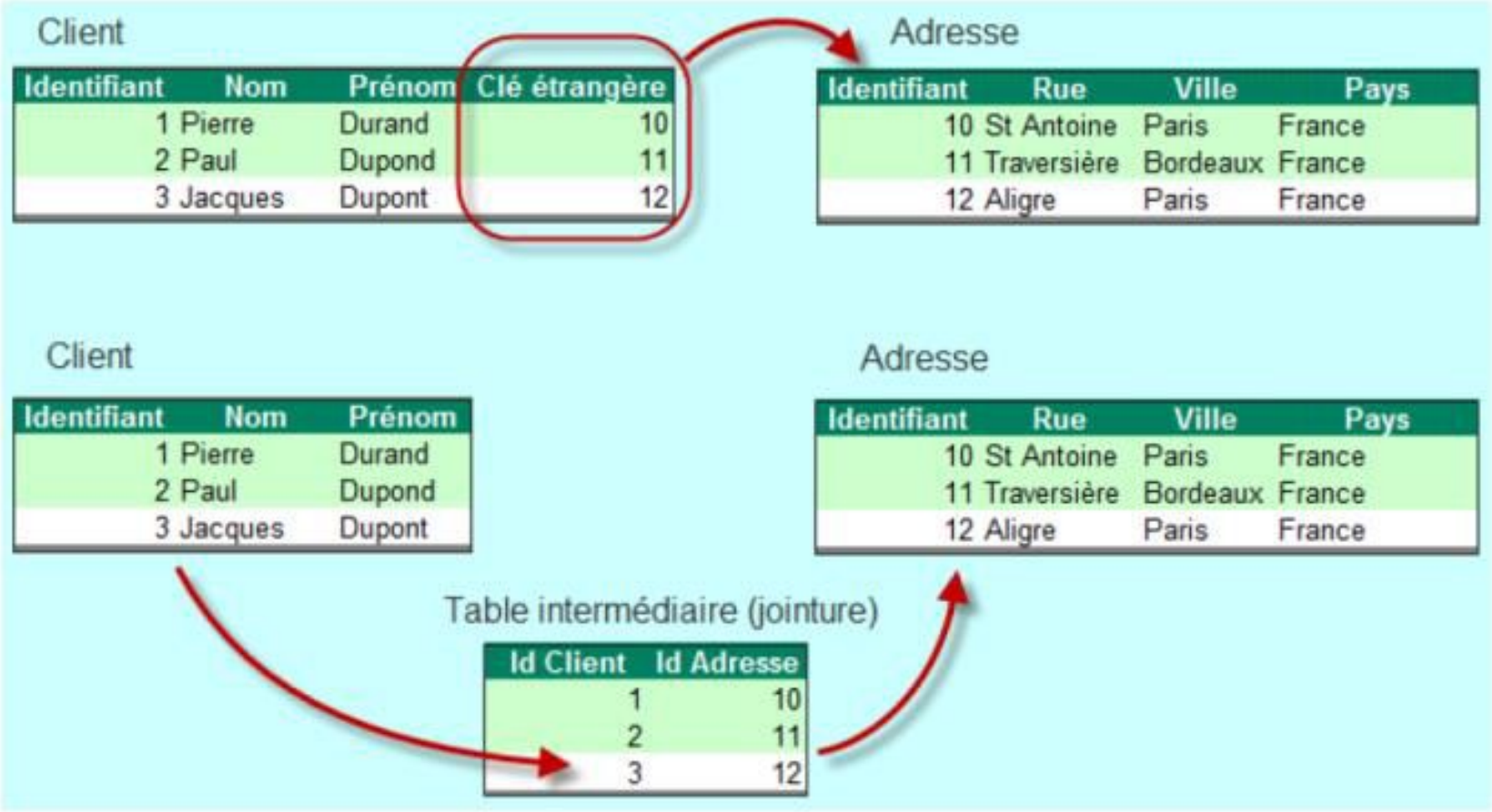
12/01/2017

Walid YAICH
walid.yaich@esprit.tn
Bureau E204

Plan

- Relations dans la base de données relationnelles
- Relations entre les entités
- Association 1 :1 unidirectionnelle
- Association 1 :1 bidirectionnelle
- Association N : 1 unidirectionnelle
- Association 1 : N unidirectionnelle
- Association N : 1 bidirectionnelle
- Association M : N unidirectionnelle
- Association M : N bidirectionnelle
- Les associations réflexives
- Association M:N porteuses de données

Relations dans la base de données relationnelles



Relations entre les entités

La plupart des entités doivent pouvoir référencer ou être en relation avec d'autres entités.

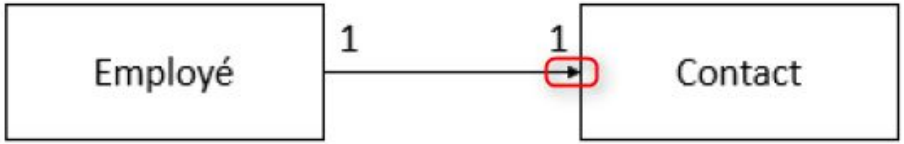
Les types d’associations entre deux entités sont :

- 1-1 : @OneToOne,
- 1-N : @OneToMany,
- N-1 : @ManyToOne,
- N-M : @ManyToMany.

Chaque type d’association peut être :

- Unidirectionnelle
- Bidirectionnelle

Association 1 :1 unidirectionnelle



L'employé possède un « Contact », alors que le « Contact » n'a aucune information sur « l'employé » auquel il est associé.

```java
@Entity
public class Employe implements Serializable {


    private Contact contact;
    @OneToOne
    public Contact getContact() {
        return contact;
    }

    public void setContact(Contact contact) {
        this.contact = contact;
    }
```

On peut changer le nom de la colonne de jointure en utilisant l'annotation **@JoinColumn**.

Association 1 :1 bidirectionnelle

C'est l'attribut « **mappedBy** » qui crée le caractère bidirectionnel de la relation et qui permet de définir les deux bouts de l'association « le maître et l'esclave », autrement, le sens de migration des clés étrangères.

Slave

```
@Entity
public class Contact implements Serializable {

    private Employe employe;

    @OneToOne(mappedBy = "contact")
    public Employe getEmploye() {
        return employe;
    }

    public void setEmploye(Employe employe) {
        this.employe = employe;
    }
```

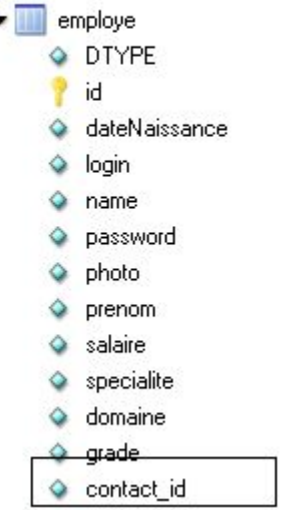
Employé: Maître

Master

```
@Entity
public class Employe implements Serializable {

    private Contact contact;

    @OneToOne
    public Contact getContact() {
        return contact;
    }

    public void setContact(Contact contact) {
        this.contact = contact;
    }
```

Contact:Esclave

Master



la classe « Employé » contient un objet de type « Contact » et le « Contact » contient un objet de type « Employé ».

Association N : 1 unidirectionnelle

L'employé possède un laboratoire alors que « le laboratoire » n'a aucune information sur « les employés » dont ils font partie.

```java
@Entity
public class Employe implements Serializable {

    private Laboratoire laboratoire;

    @ManyToOne
    public Laboratoire getLaboratoire() {
        return laboratoire;
    }

    public void setLaboratoire(Laboratoire laboratoire) {
        this.laboratoire = laboratoire;
    }
```

Association 1 : N unidirectionnelle

Le laboratoire possède une collection d'employés alors que « l'employé » n'a aucune information sur « le laboratoire » auquel il appartient

```
public class Laboratoire implements Serializable {

    private List<Employe> employes;
    @OneToMany
    public List<Employe> getEmployes() {
        return employes;
    }
    public void setEmployes(List<Employe> employes) {
        this.employes = employes;
    }
```

Association N : 1 bidirectionnelle

L'attribut mappedBy est défini pour l'annotation @OneToMany, mais pas pour l'annotation @ManyToOne.

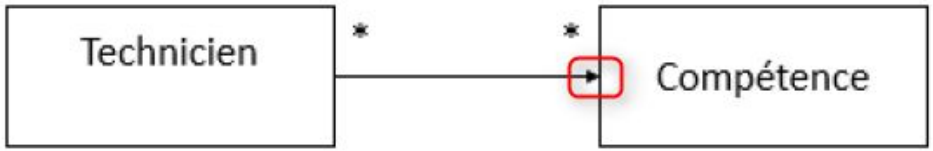
Slave

```
@Entity
public class Laboratoire implements Serializable {

    private List<Employe> employes;

    @OneToMany(mappedBy="laboratoire")
    public List<Employe> getEmployes() {
        return employes;
    }

    public void setEmployes(List<Employe> employes) {
        this.employes = employes;
    }
```

Master

```
@Entity
public class Employe implements Serializable {
    private Laboratoire laboratoire;

    @ManyToOne
    public Laboratoire getLaboratoire() {
        return laboratoire;
    }

    public void setLaboratoire(Laboratoire laboratoire)
        this.laboratoire = laboratoire;
    }
```

Association M : N unidirectionnelle



Le technicien possède une « collection de compétences » alors que « la compétence » n'a aucune information sur « les techniciens » auquel elle est associé.

```
public class Technicien extends Employe implements Serializable {

    private List<Competence> competences;

    @ManyToMany

    public List<Competence> getCompetences() {
        return competences;
    }
    public void setCompetences(List<Competence> competences) {
        this.competences = competences;
    }
```

Association M : N bidirectionnelle

Le Mapping de cette relation dans la base de données génère une table associative qui contient les deux clés étrangères des deux entités.

Slave

```
@Entity
public class Technicien extends Employe implements Serializable {

    private List<Competence> competences;

    @ManyToMany (mappedBy="techniciens")

    public List<Competence> getCompetences() {
        return competences;
    }
    public void setCompetences(List<Competence> competences) {
        this.competences = competences;
    }
```

Competence : master

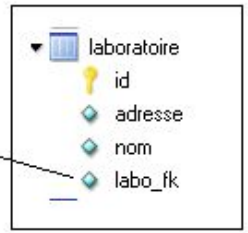
Master

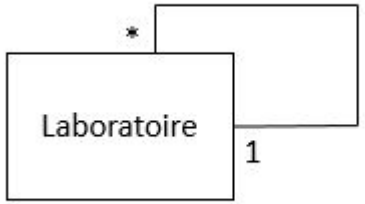
```
@Entity
public class Competence implements Serializable {

    private List<Technicien> techniciens;

    @ManyToMany
    public List<Technicien> getTechniciens() {
        return techniciens;
    }

    public void setTechniciens(List<Technicien> techniciens) {
        this.techniciens = techniciens;
    }
```

technicien: Slave

Les associations réflexives

```java
@Entity
public class Laboratoire implements Serializable {

    private Laboratoire labo;

    private List<Laboratoire> miniLabos;

    @ManyToOne
    @JoinColumn(name="labo_fk")
    public Laboratoire getLabo() {
        return labo;
    }

    public void setLabo(Laboratoire labo) {
        this.labo = labo;
    }

    @OneToMany(mappedBy = "labo", fetch = FetchType.EAGER)

    public List<Laboratoire> getMiniLabos() {
        return miniLabos;
    }

    public void setMiniLabos(List<Laboratoire> miniLabos) {
        this.miniLabos = miniLabos;
    }
```





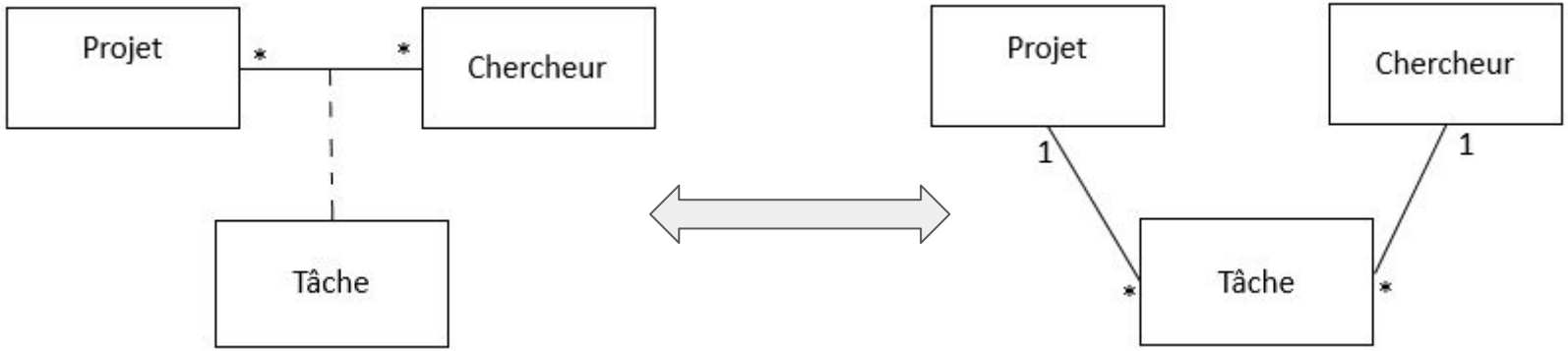
Une association réflexive est une association qui relie des occurrences de la même entité, elle peut être de type 1:1, 1:N, N:1, M:N.

Association M:N porteuses de données

Une association ManyToMany est dite porteuse de données si la classe associative comporte des données autres que sa clé primaire.

Exemple :

Une tâche est caractérisée par un projet, un chercheur, un nom et une durée.



Association M:N porteuses de données

Définition de la clé primaire

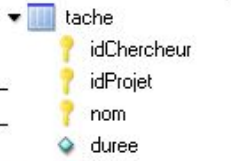
```java
@Embeddable
public class TachePk implements Serializable {

    private int idChercheur;
    private int idProjet;
    private String nom;
```

```java
@Entity
public class Tache implements Serializable {

    private TachePk tachePk;
    private int duree;

    @EmbeddedId
    public TachePk getTachePk() {
        return tachePk;
    }
```

Association M:N porteuses de données

```java
@Entity
public class Tache implements Serializable {

    private Projet projet;
    private Chercheur chercheur;

    @ManyToOne
    @JoinColumn(name="idProjet",referencedColumnName="id"
    ,insertable=false,updatable=false)
    public Projet getProjet() {
        return projet;
    }

    public void setProjet(Projet projet) {
        this.projet = projet;
    }

    @ManyToOne
    @JoinColumn(name="idChercheur",referencedColumnName="id"
    ,insertable=false,updatable=false)
    public Chercheur getChercheur() {
        return chercheur;
    }
```

```java
@Entity

public class Chercheur extends Employe implements Serializable {

    private List<Tache> taches;
    @OneToMany(mappedBy="chercheur")
    public List<Tache> getTaches() {
        return taches;
    }
    public void setTaches(List<Tache> taches) {
        this.taches = taches;
    }
```




```java
@Entity
public class Projet implements Serializable {

    private List<Tache> taches;

    @OneToMany(mappedBy = "projet")
    public List<Tache> getTaches() {
        return taches;
    }

    public void setTaches(List<Tache> taches) {
        this.taches = taches;
    }
```