

BASES DE DONNEES ORIENTEES OBJET

LE RELATIONNEL 'OBJET'

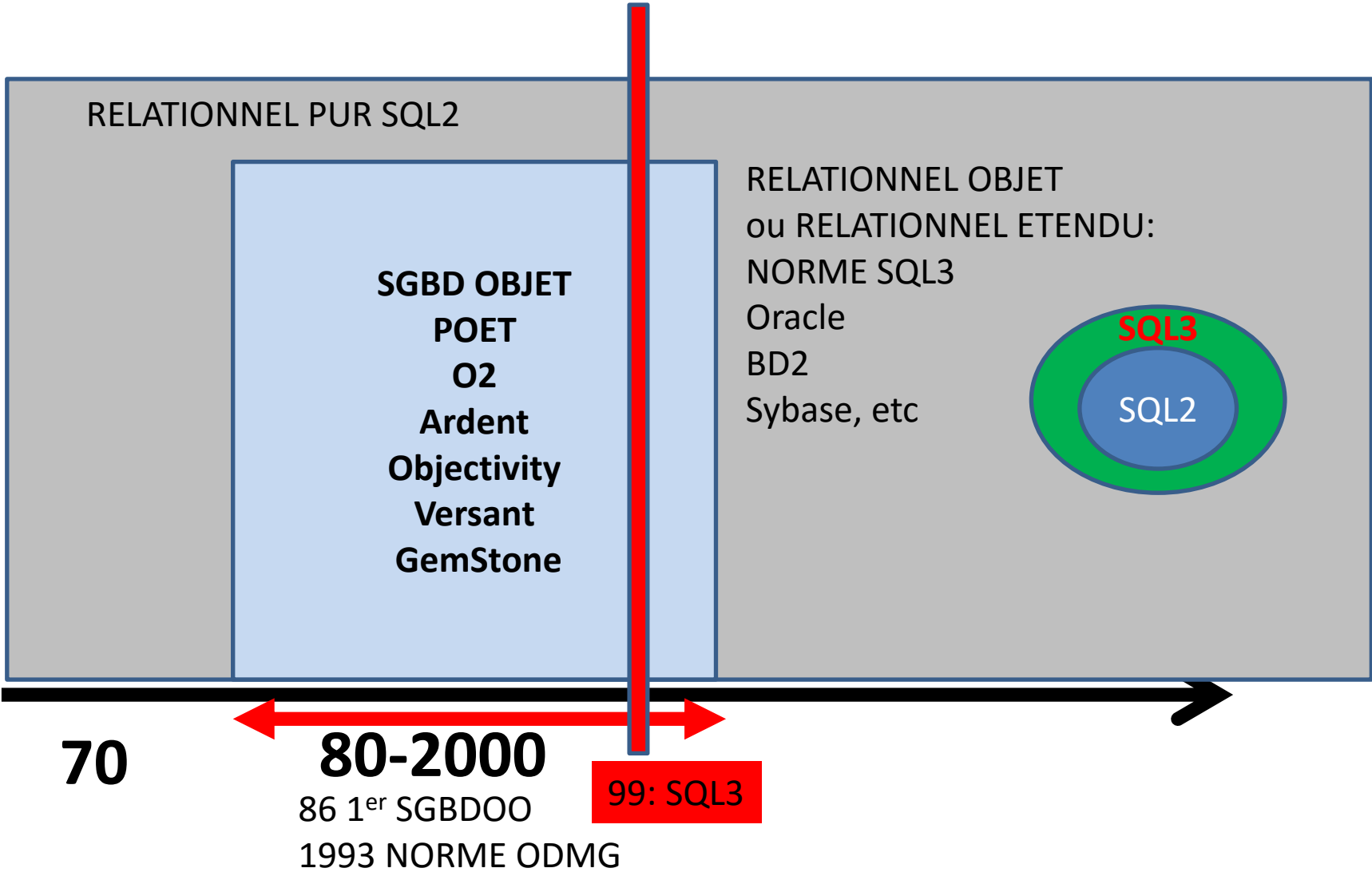
OBJECTIF DE CE COURS

- FAIRE DES BASES DE DONNEES OBJETS:
 - L'ELEMENT DE STOCKAGE DES DONNEES DE LA BASE EST :

L'OBJET



UNE PETITE HISTOIRE



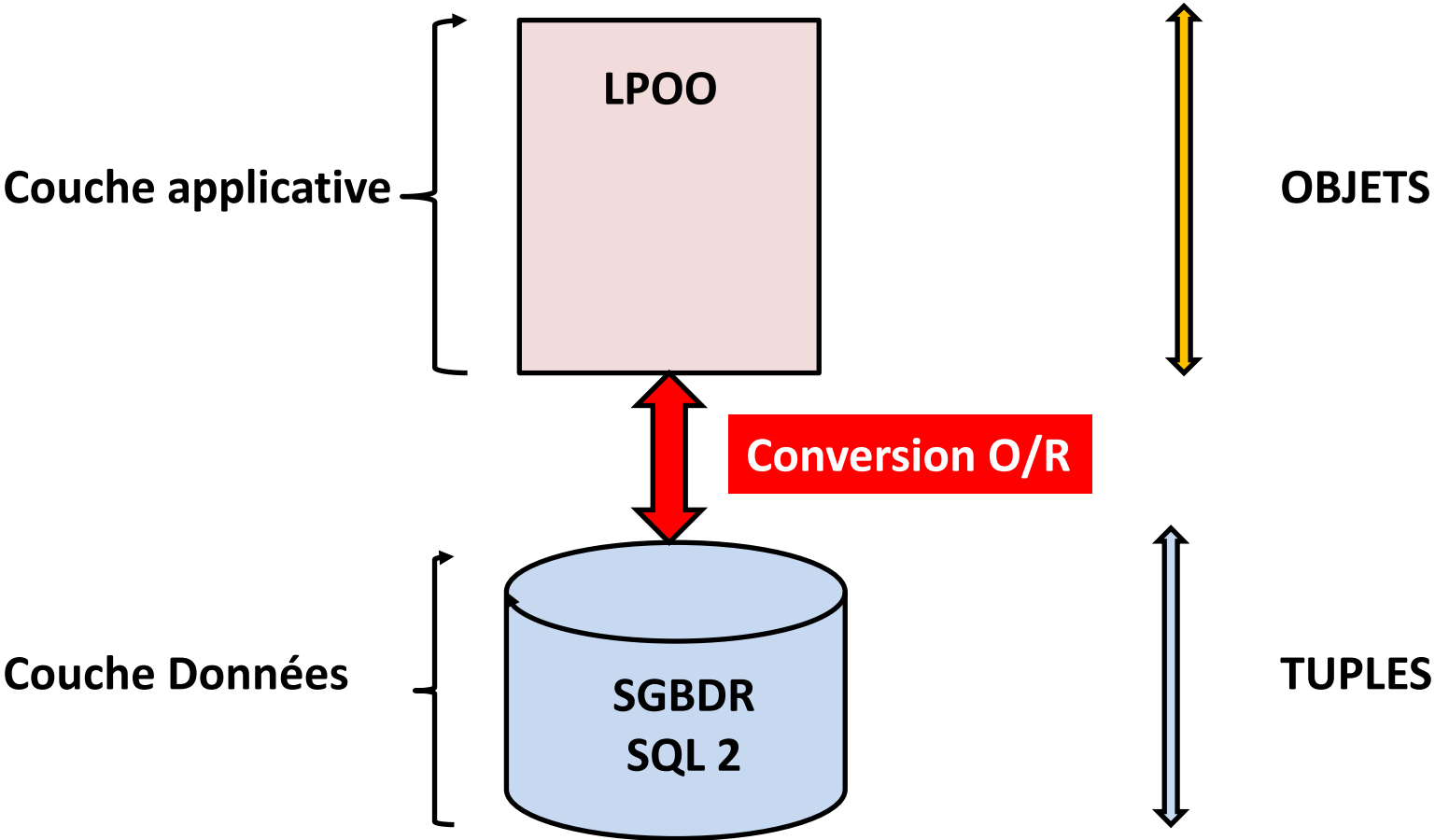
MOTIVATION

PRINCIPE DES BASES DE DONNEES:

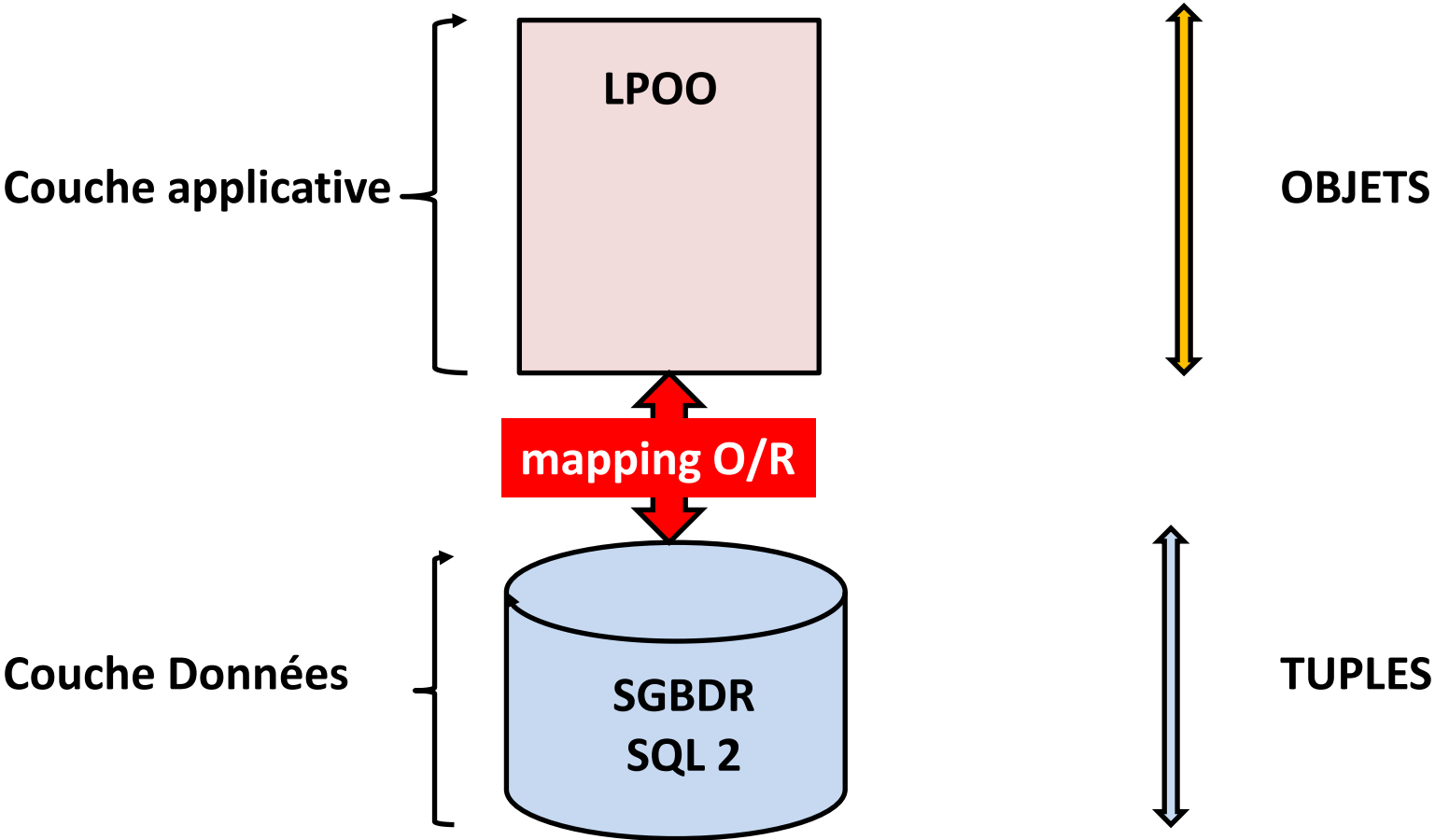


PRINCIPE DES BASES DE DONNEES

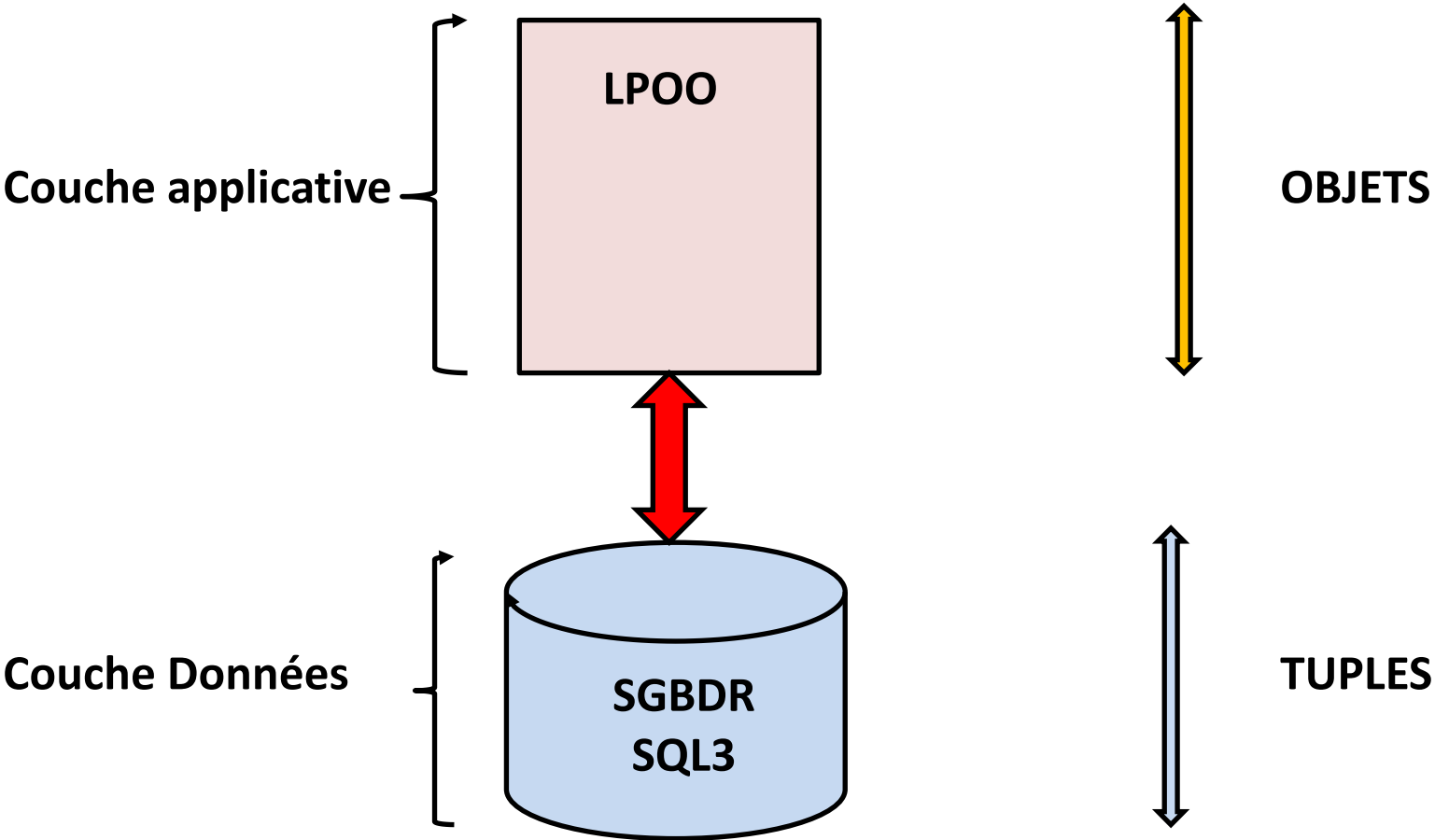
Introduction: Pourquoi les SGBDOO???



Introduction: Pourquoi les SGBDOO???



Introduction: Pourquoi les SGBDOO???



PRE-REQUIS

- **BASES DE DONNEES RELATIONNELLES**
- **PARADIGME OO**
- **LPOO**

- **SQL**
- **PL/SQL**
- **JAVA**

BIBLIOGRAPHIE

- 1. Introduction aux bases de données,
Chris J. Date, Vuillebert,
6me Edition, 1998.**
- 2. Bases de données objet et relationnel,
G. Gardarin,
Eyrolles, 1999**
- 3. Les objets, M. Bouzeghoub,
G. Gardarin, P. Valduriez,
Eyrolles, 1998**
- 4. Objet-relationnel sous Oracle8,
C. Soutou,
Eyrolles, 1999**
- 5. WWW.OMG.ORG**

SOMMAIRE GENERAL

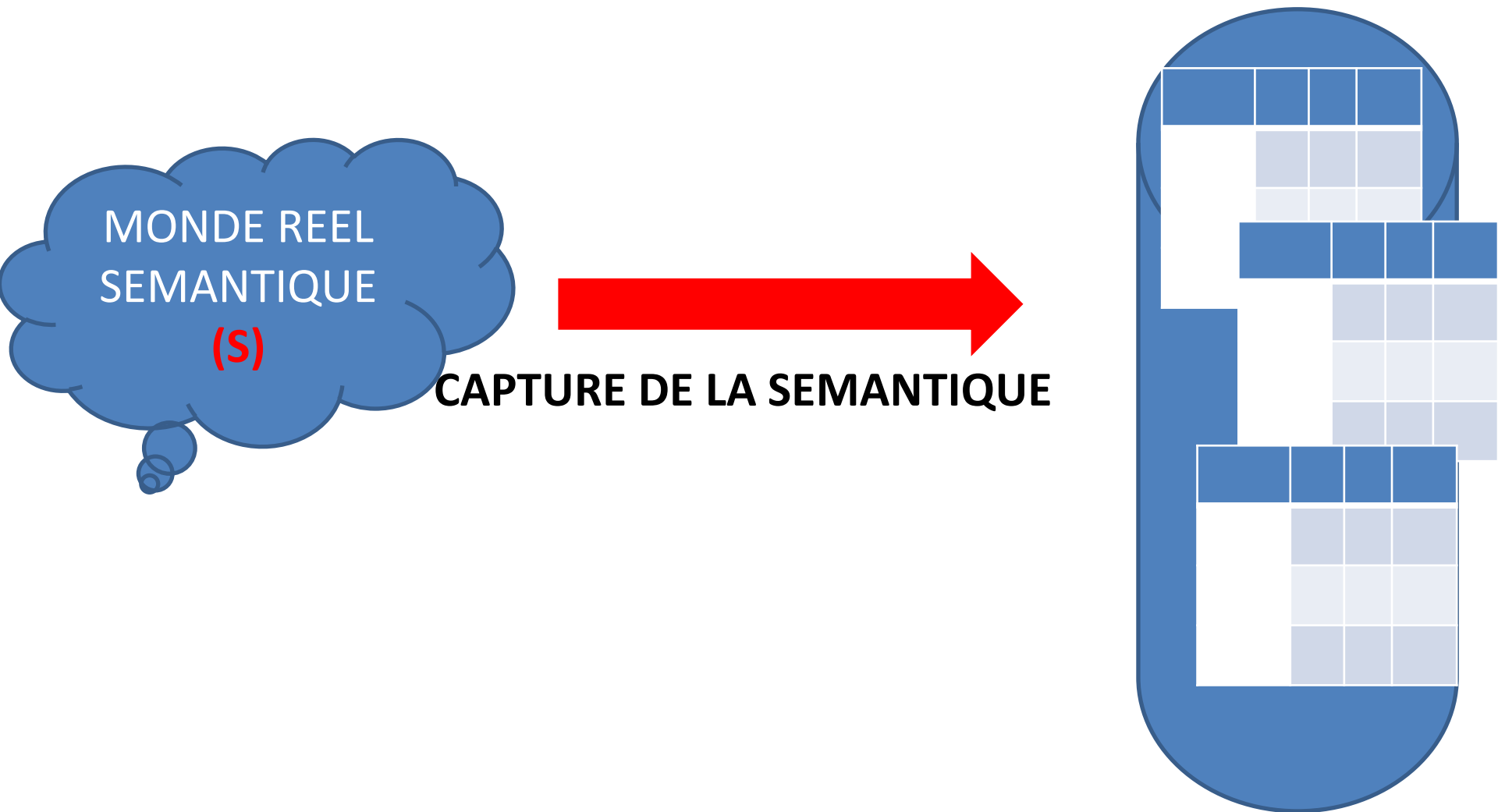
- **I. Introduction : pourquoi les bases de données objet**
- **II. L'apport des SGBDOO**
- **III. Le relationnel étendu**
- **IV. Manipulation du relationnel étendu: Oracle (TP)**
- **V. Conclusion**

- **I. Introduction : pourquoi les bases de données objet**
- **II. L'apport des SGBDOO**
- **III. Le relationnel étendu**
- **IV. Manipulation du relationnel étendu: Oracle (TP)**
- **V. Conclusion**

CRITIQUE DU MODELE RELATIONNEL

**PRENONS UN PEU DE REcul
VIS-À-VIS DU
RELATIONNEL!!!!!!**

PRINCIPE DES BASES DE DONNEES:



LIMITES DU MODELE RELATIONNEL : structures de données

PERSONNES	NUMP	NOM	PRENOM	DTNAIS	PERE	MERE	CONJOINT
	111	toto	titi		325	125	333
	325						
	125						
	896				111	333	
	333	tata					111

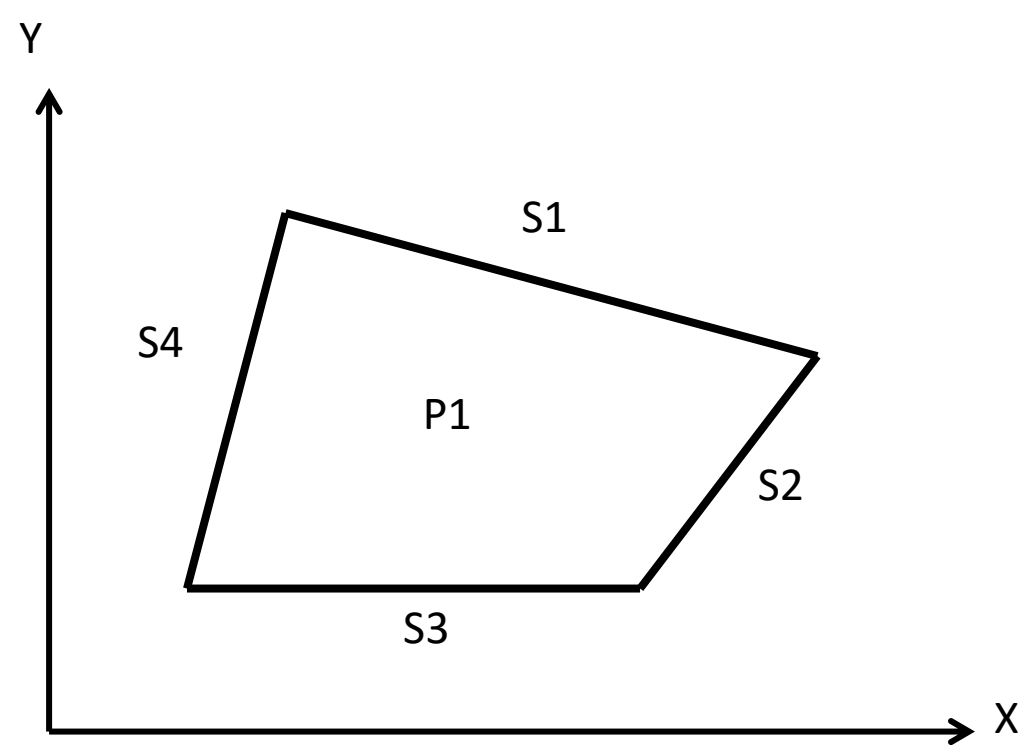
Structures de données élémentaires:
Table, ligne.

Type de données atomiques élémentaires:
NUMBER, VARCHAR, CHAR, DECIMAL, DATE, TIME ,ETC.



∀ La sémantique du monde réel modélisé, il faut pouvoir l'exprimer avec ces structures du modèle relationnel

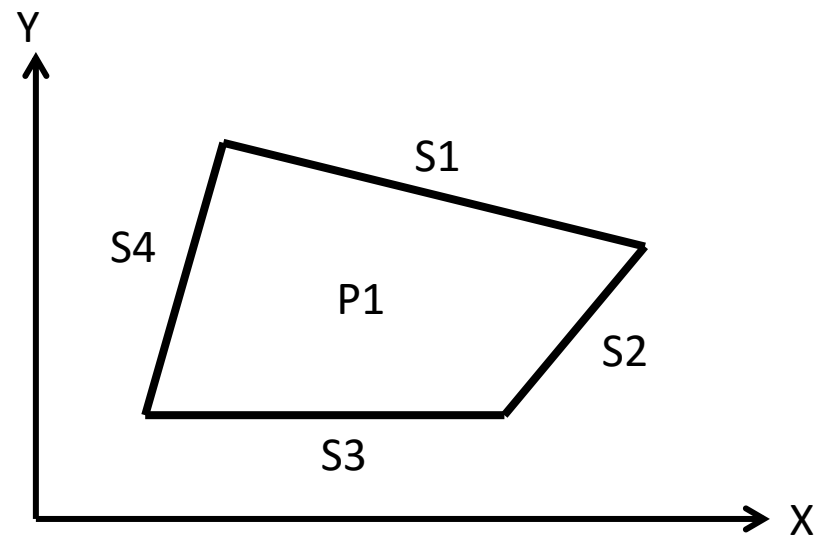
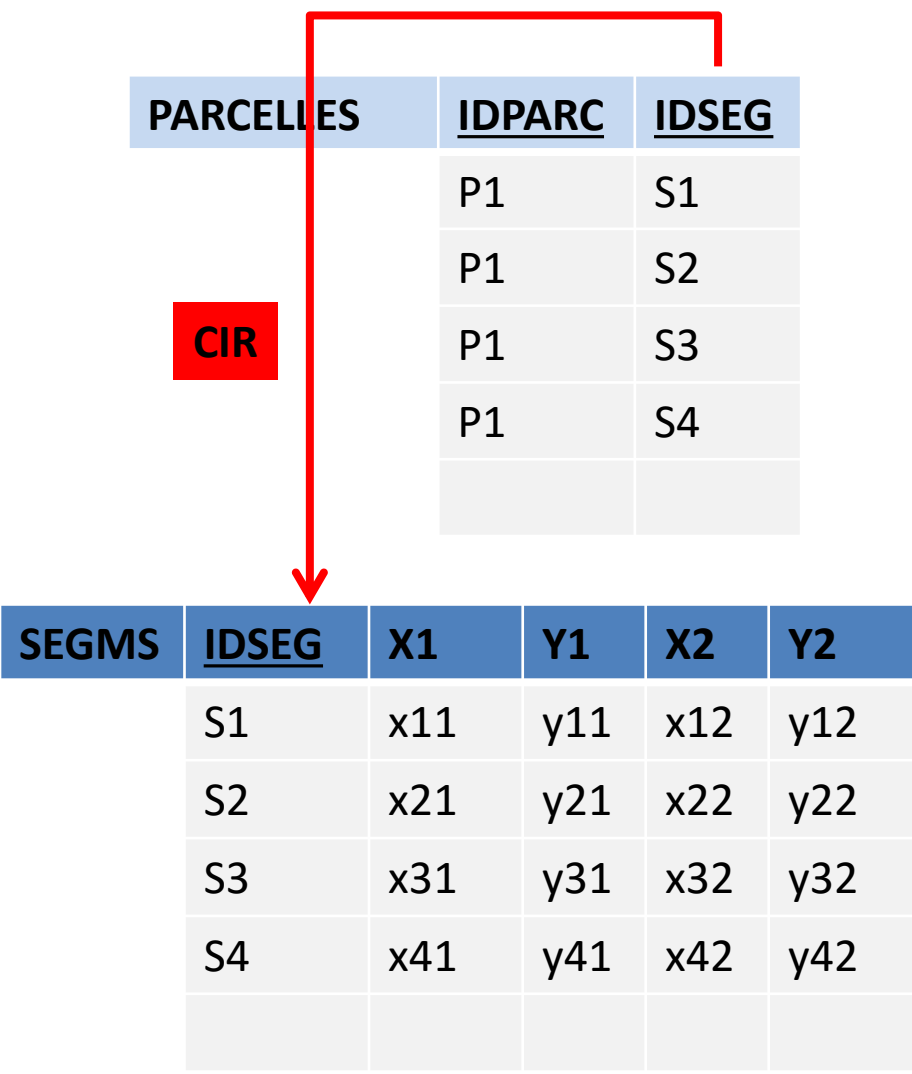
LIMITES DU MODELE RELATIONNEL : Capture de la sémantique



ENTITE SEMANTIQUE ELEMENTAIRE DANS LE MONDE MEDELISE: PARCELLE DE TERRAIN

COMMENT IMPLEMENTER CETTE ENTITE DANS LE MODELE RELATIONNEL???

LIMITES DU MODELE RELATIONNEL : Capture de la sémantique



2 TABLES ET
UNE CONTRAINTE D'INTEGRITE REFERENTIELLE
ET
APRES????????????????!!!!!!!!!!!!!!

LIMITES DU MODELE RELATIONNEL : Capture de la sémantique

Le modèle relationnel SUBDIVISE une entité sémantique du monde réel en plusieurs morceaux et stocke chaque morceau dans des tables différentes



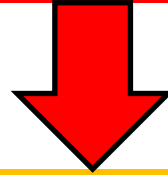
La contrainte d'intégrité référentielle pour garder le lien entre les morceaux de la même entité sémantique



L'opérateur de jointure pour "RECOLLER LES MORCEAUX"

**ET LA SEMANTIQUE DANS TOUT CA??
LE RELATIONNEL EST-IL CAPABLE DE ME RESTITUER TOUTES LA SEMANTIQUE AVEC
SES OUTILS FOURNIS???**

La surface de la parcelle P1???



```
SELECT S.IDSEG, S.X1, S.Y1, S.X2, S.Y2  
FROM SEGMS S, PARCELLES P  
WHERE S.IDSEG=P.IDSEG AND P.IDPAR=P1;
```

AVEC SQL LE RELATIONNEL N'EST MEME PAS CAPABLE DE RESTITUER LA SEMANTIQUE DU DEPART!!!!



IL FAUT UN AUTRE LANGAGE DE PROGRAMMATION (PL)

SQL est un langage INCOMPLET DANS LE SENS DE LA CALCULABILITE

DONC, POUR FAIRE LES BD, IL FAUT MAITRAISER AU MOINS 2 LANGAGES:

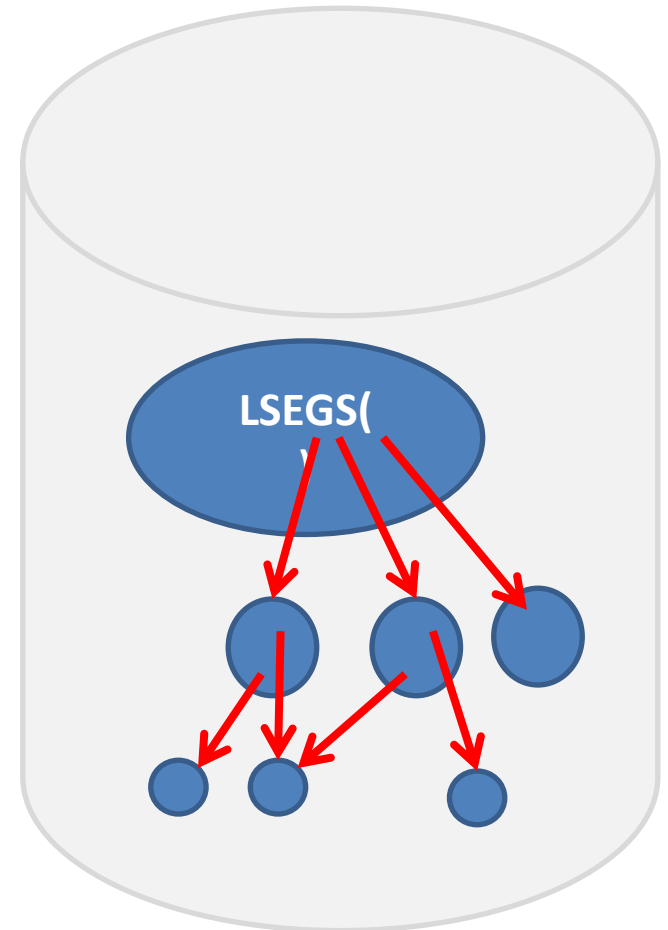
- **SQL POUR LES ACCES AUX DONNEES DANS LA BASE**
- **UN AUTRE LANGAGE DE PROGRAMMATION (PL) POUR PALIER AUX INSUFFISANCES DE SQL**

LIMITES DU MODELE RELATIONNEL : Capture de la sémantique

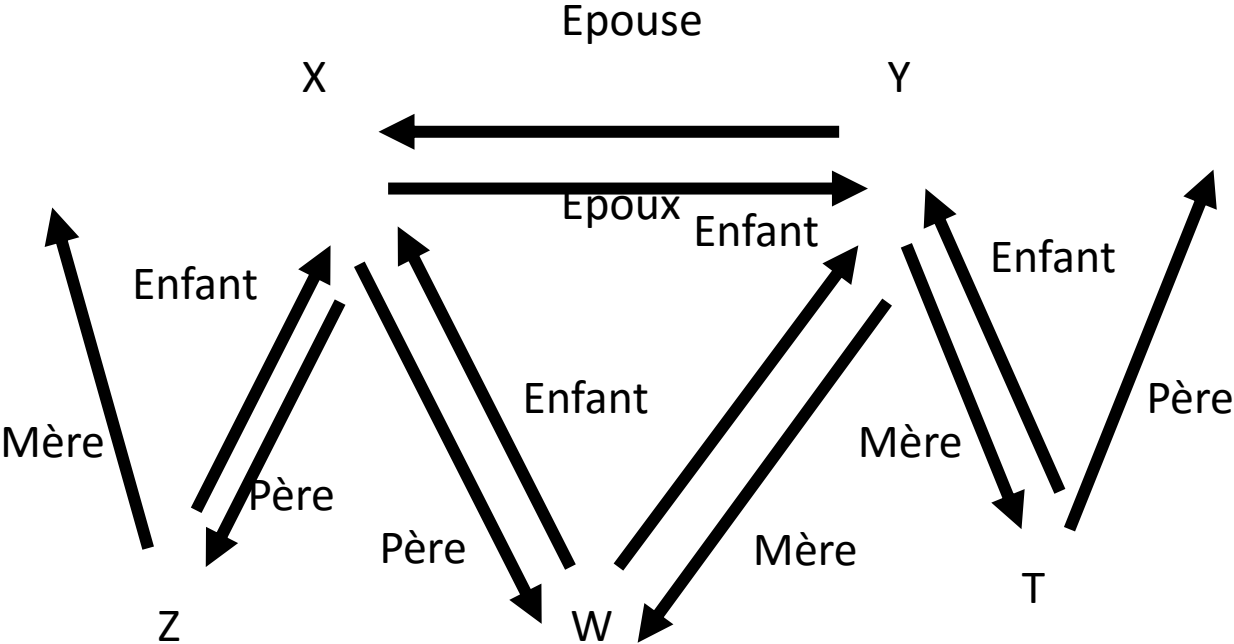
```
Classe CPARCELLES{  
    private int          IDPARC;  
    private Liste<CSEGMS> LSEGS;  
    .....  
}
```

```
Classe CSEGMS{  
    private int          IDSEG;  
    private CPOINTS      PTS1;  
    private CPOINTS      PTS2;  
    ..... }  
}
```

```
Classe CPOINT{  
    private int          IDPOINT;  
    private int          X;  
    private int          Y;  
    .....  
}
```



LIMITES DU MODELE RELATIONNEL : Capture de la sémantique



LIMITES DU MODELE RELATIONNEL : Capture de la sémantique

PERSONNES	NUMP	NOM	PRENOM	DTNAIS	PERE	MERE	CONJOINT
	111				325	125	333
	325						
	125						
	896				111	333	
	333						111

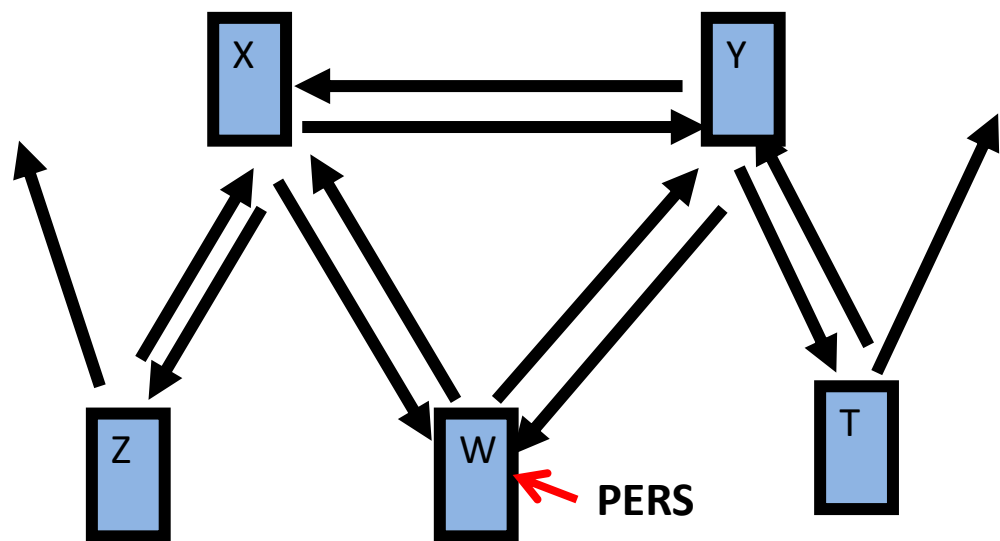
```
graph TD
    111 --> 325
    111 --> 125
    111 --> 333
    325 --> 125
    125 --> 896
    896 --> 333
    333 --> 111
```

Les frères et sœurs (à part entière) de la personne numéro 896??

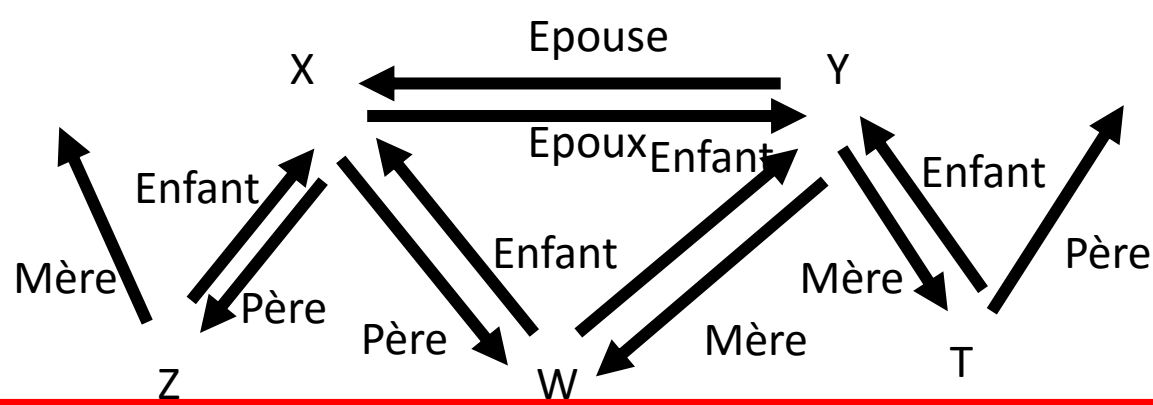
```
SELECT P1.NUMP
FROM PERSONNES P1, PERSONNES P2
WHERE P1.PERE=P2.PERE AND P1.MERE=P2.MERE AND P2.NUMP=896;
```


LIMITES DU MODELE RELATIONNEL : Capture de la sémantique

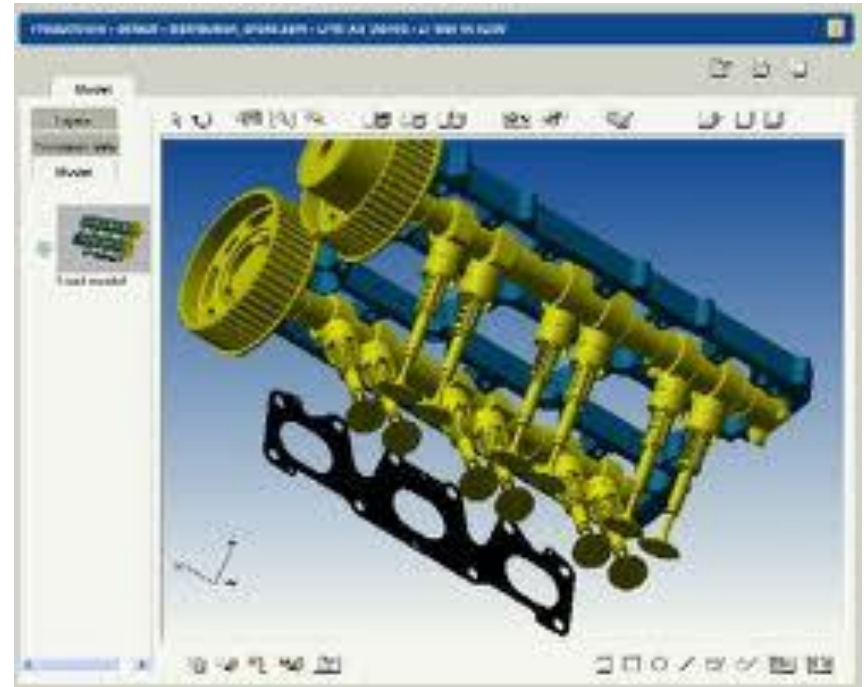
```
Class Personne{
private Integer      NUMP;
private String       NOM;
private String       PRENOM;
private Date         DTNAIS;
private Personne     PERE;
private Personne     MERE;
private Personne     CONJOINT;
private List<Personne> ENFANTS;
.....
.....
}
```



PERS.PERE.getEnfants() ∩ PERS.MERE.getEnfants();



LIMITES DU MODELE RELATIONNEL : NOUVELLES APPLICATIONS



LIMITES DU MODELE RELATIONNEL : CONCLUSION

Insuffisance du modèle relationnel

Pauvreté des structures de données (tuple, table, integer, ...)

Faible capture de la sémantique (intégrité référentielle)

Incomplétude du SQL

Problème SQL/langage de programmation

Nouveaux besoins

Nouveaux types de données (image, dessin, vidéo, spatial..)

Nouvelles applications (bd multimédia, bd spatiale, etc.)

SOMMAIRE GENERAL

- I. Introduction : pourquoi les bases de données objet
- **II. L'apport des SGBDOO**
- III. Le relationnel étendu
- IV. Manipulation du relationnel étendu: Oracle (TP)
- V. Conclusion

DEFINITION D'UN SGBDOO (la norme)

DEFINITION D'UN SGBDOO : NORME OMG

Aspect Bases de Données

- **Persistence**
- **Gestion de mémoires**
- **Langage de requêtes**
- **Multi-utilisateurs**
- **Fiabilité**
- **Sécurité**

Aspect Orienté Objet

- **Identité d'objet**
- **Objet complexe**
- **Encapsulation**
- **Type ou Classe**
- **Héritage**
- **Surcharge et résolution tardive**
- **Complétude**
- **Extensibilité**

DEFINITION D'UN SGBDOO : NORME OMG

Les règles facultatives

- Héritage multiple
- Généricité
- Vérification de type
- Distribution
- Transaction de conception
- Versions

Les règles ouvertes

- Modèle de calcul
- Modèle de persistance
- Uniformité
- Nommage

DEFINITION D'UN SGBDOO : NORME OMG

Aspect Bases de Données

- **Persistence**
- **Gestion de mémoires**
- **Langage de requêtes**
- **Multi-utilisateurs**
- **Fiabilité**
- **Sécurité**

DEFINITION D'UN SGBDOO

Aspect Orienté Objet

- **Identité d 'objet**
- **Objet complexe**
- **Encapsulation**
- **Type ou Classe**
- **Héritage**
- **Surcharge et résolution tardive**
- **Complétude**
- **Extensibilité**

**IDENTITE D'OBJET
et
PERSISTANCE**

Problème???

La persistance=les objets survivent à l'application qui les a créés.

La persistance est à la charge du SGBD.

Comment reconnaître les objets d'une façon unique est sûr???

Comment rendre les objets persistants?

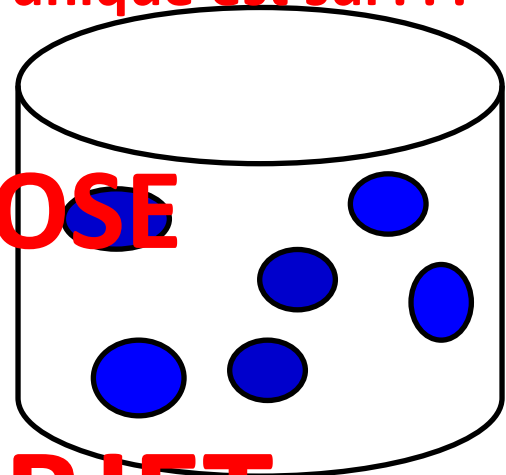
~~La valeur de l'objet?~~

IL FAUT AUTRE CHOSE

~~L'adresse physique de l'objet?~~

L'IDENTITE DE L'OBJET

~~Notion de clé~~



Disque Dur

- Les objets sont permanents dans la BD
- C'est le SGBD qui gère les objets dans la BD

- Chaque objet a une identité (**OID**) indépendante de:

- Sa valeur
- Son type ou classe
- Sa localisation physique



- L'identité d'un objet est:

- UNIQUE
- IMMuable
- PERMANENTE

OBJET=<OID, VALEUR>

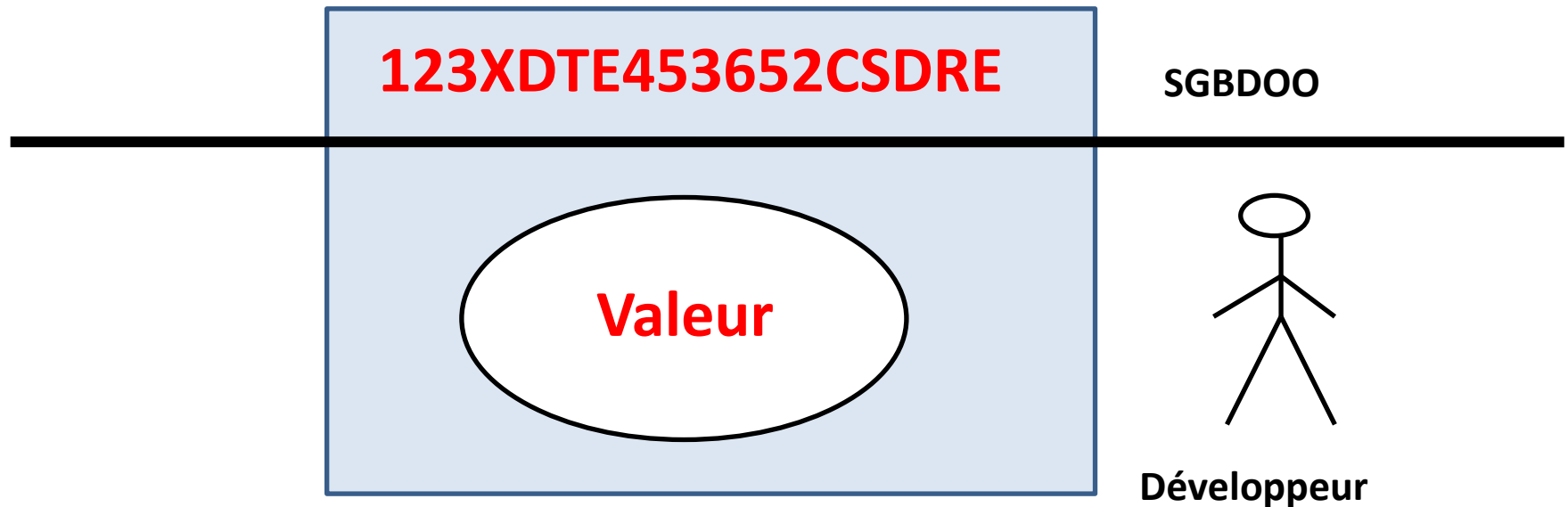
- L'identité d'un objet est attribuée par le SGBD et gérée uniquement par le SGBD

- L'identité d'objet n'est pas accessible par le développeur

La base de données=les valeurs des objets

Le développeur voit la base de données=les valeurs des objets

Le SGBDOO voit les objets=les OID des objets



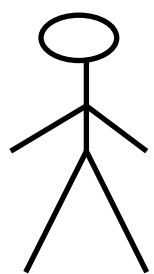
Quelles sont les conséquences sur notre façon de faire les BD????

Conséquence 1: deux visions

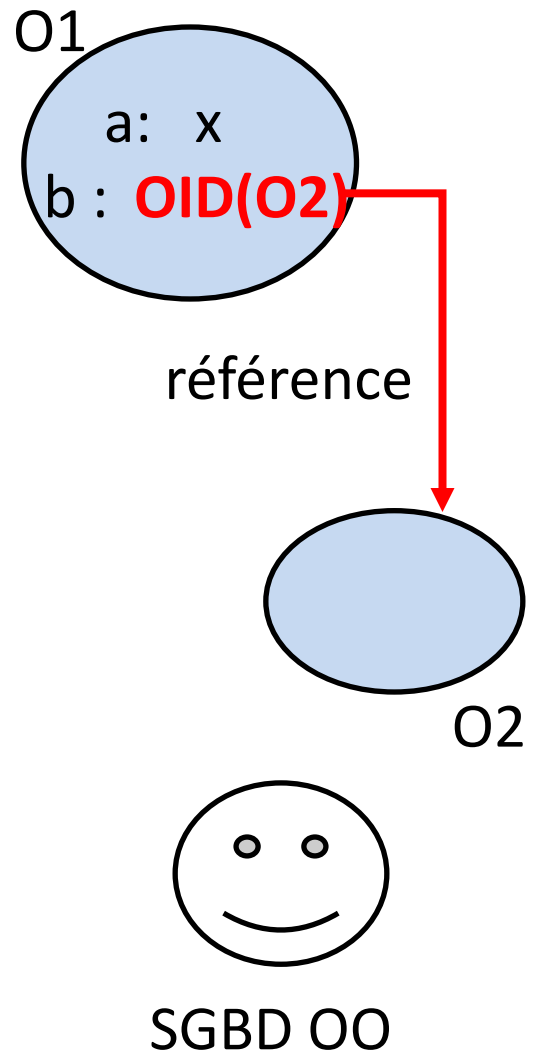
Class A (a : type1,
b : **B**)

Class B (c : type2,
d : type3)

```
A O1 : new A,  
B O2 : new B  
O1.a:=x;  
O1.b:=O2;
```



Développeur



Conséquence: deux visions

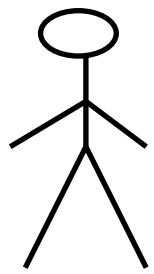
Class A (a : type1,
b : **B**)

Class B (c : type2,
d : type3)

PAS DE CONTRAINTNE D'INTEGRITE
REFERENTIELLE

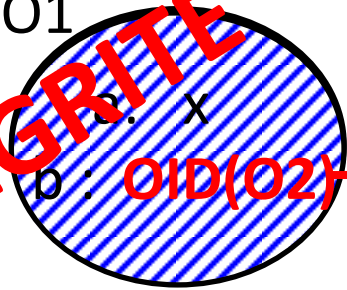
O1.a

O1.b

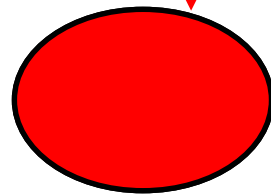


Développeur

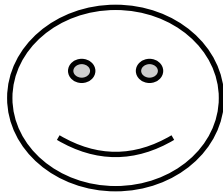
O1



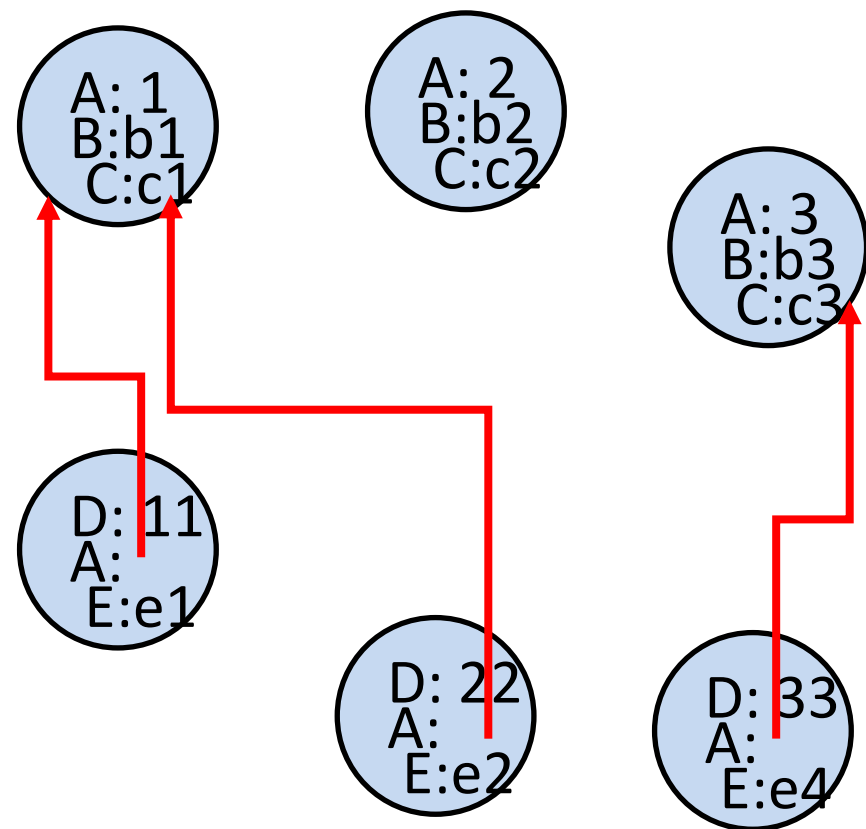
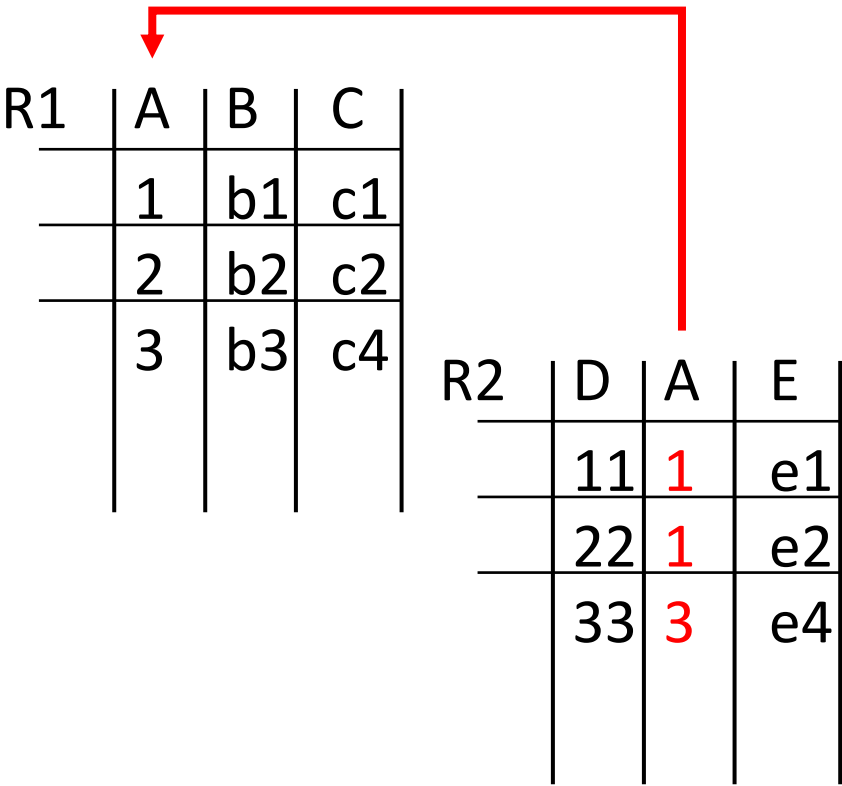
référence



O2



SGBD OO



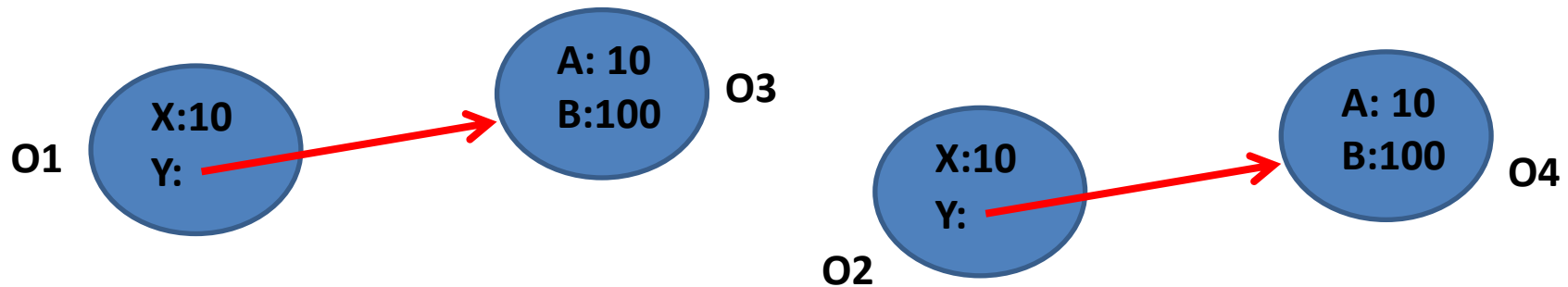
Modèle relationnel: système à valeur

Modèle objet: système à objet

IDENTITE ET EGALITE

La sémantique de la BD est dans les valeurs des objets

Les objets peuvent contenir des références vers d'autres objets



- Test d'identité ==
- Test d'égalité « de surface » =
- Test d'égalité « profonde » =* (deep equality)
(égalité arborescente)

IDENTITE ET EGALITE

Class A (a : entier, b : B)
Class B (c : entier, d : entier)

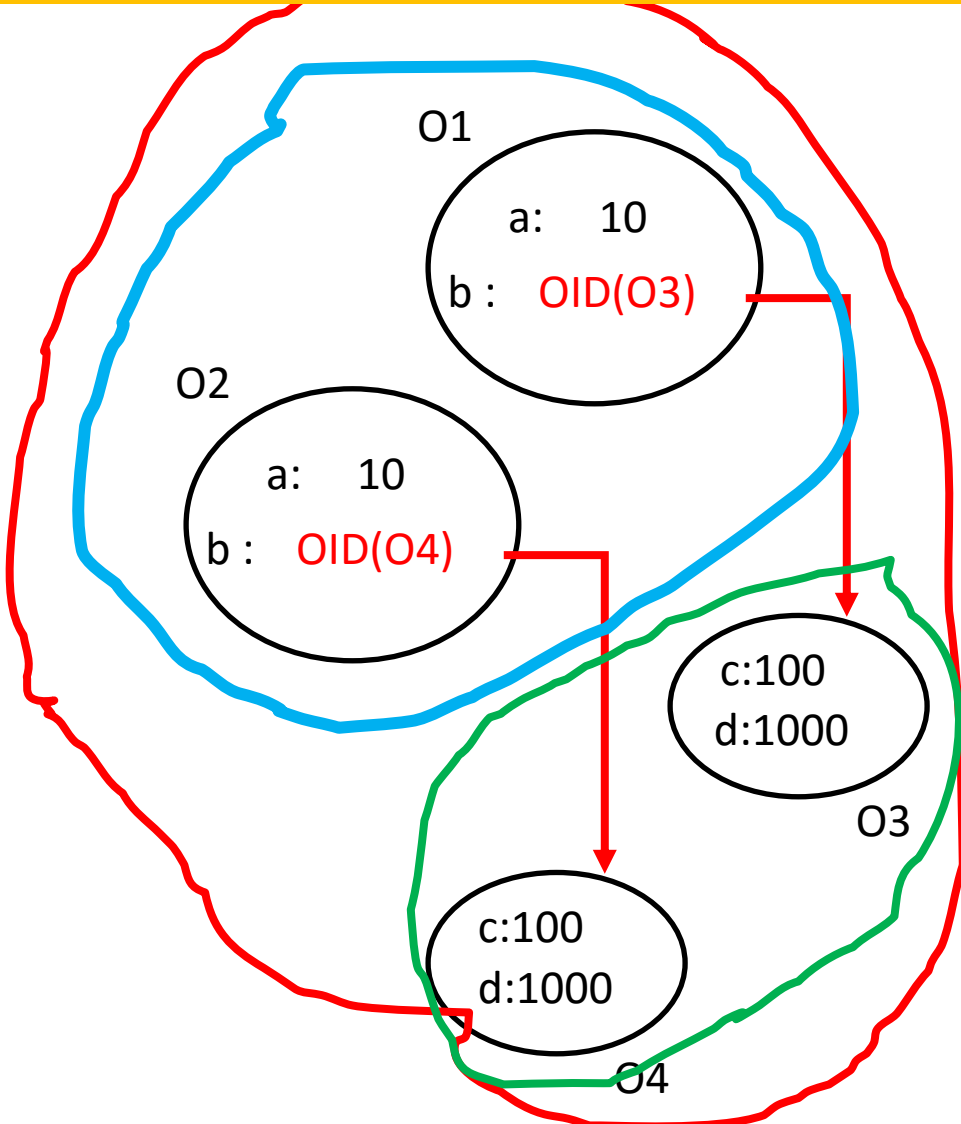
```
A O1 : new A;
A O2 : new A;
B O3 : new B;
B O4 : new B;

O2.a:=10;
O2.b:=O4;

O3.c:=100;
O3.d:=1000;

O4.c:=100;
O4.d:=1000;

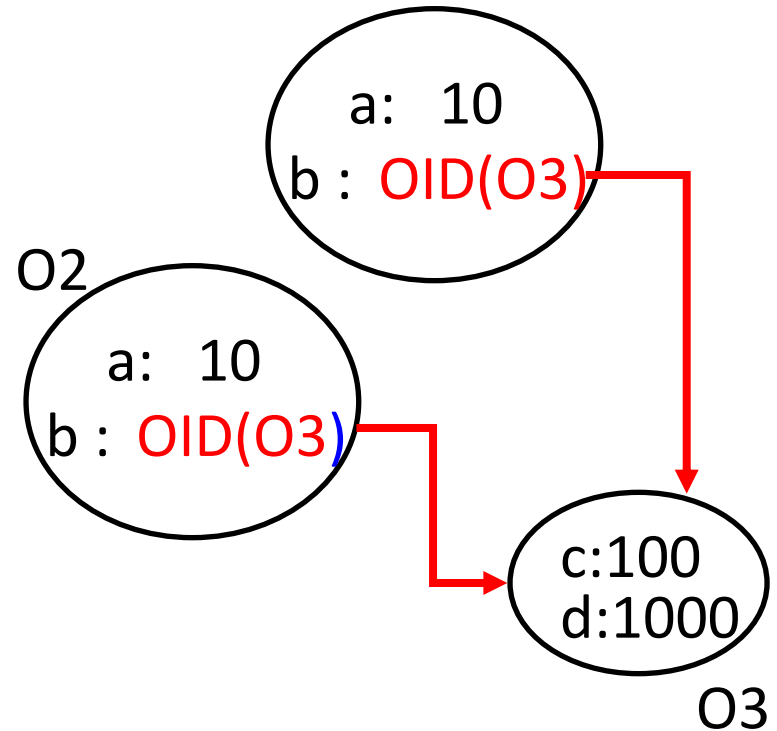
O1.a:=10;
O1.b:=O3;
```



IDENTITE ET EGALITE

Class A (a : entier, b : **B**)
 Class B (c : entier, d : entier)

A O1 : new A;	O2.a:=10;
A O2 : new A;	O2.b:= O3 ;
B O3 : new B;	
B O4 : new B;	O1=O2?
O3.c:=100;	
O3.d:=1000;	O1=*O2?
O1.a:=10;	
O1.b:= O3 ;	



IDENTITE ET EGALITE

Class A (a : entier, b : entier)

A O1 : new A;

A O2;

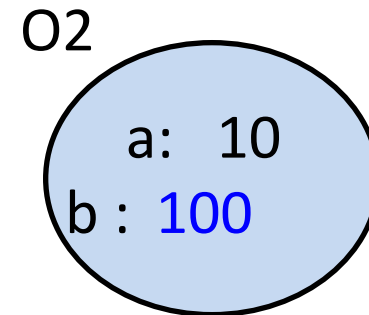
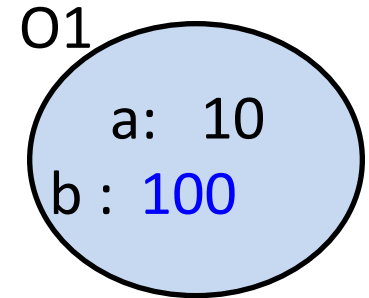
O1.a:=10;

O1.b:=100;

O2:=copy(O1)

O2==O1?

O2=O1?



Un objet est CLONE et NON COPIE

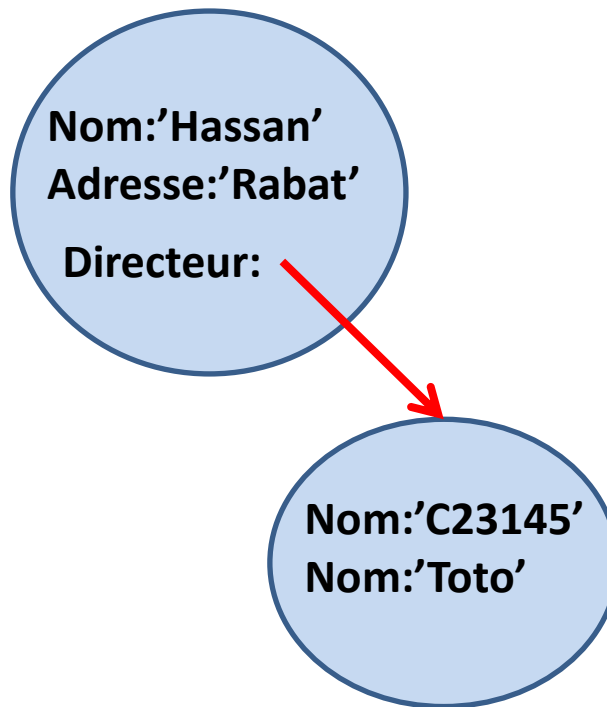
IDENTITE ET PARTAGE D'OBJET

```
class Personne{  
private Integer    CIN;  
private String    Nom;  
.....  
}
```

```
class Hotel {  
private String    Nom;  
private String    Adresse;  
Private  Personne Directeur;  
.....  
}
```

```
Personne pers=new Personne('C23145','toto');  
Hotel Hot=new Hotel('Hassan','Rabat');
```

```
Hot.Directeur=pers;
```



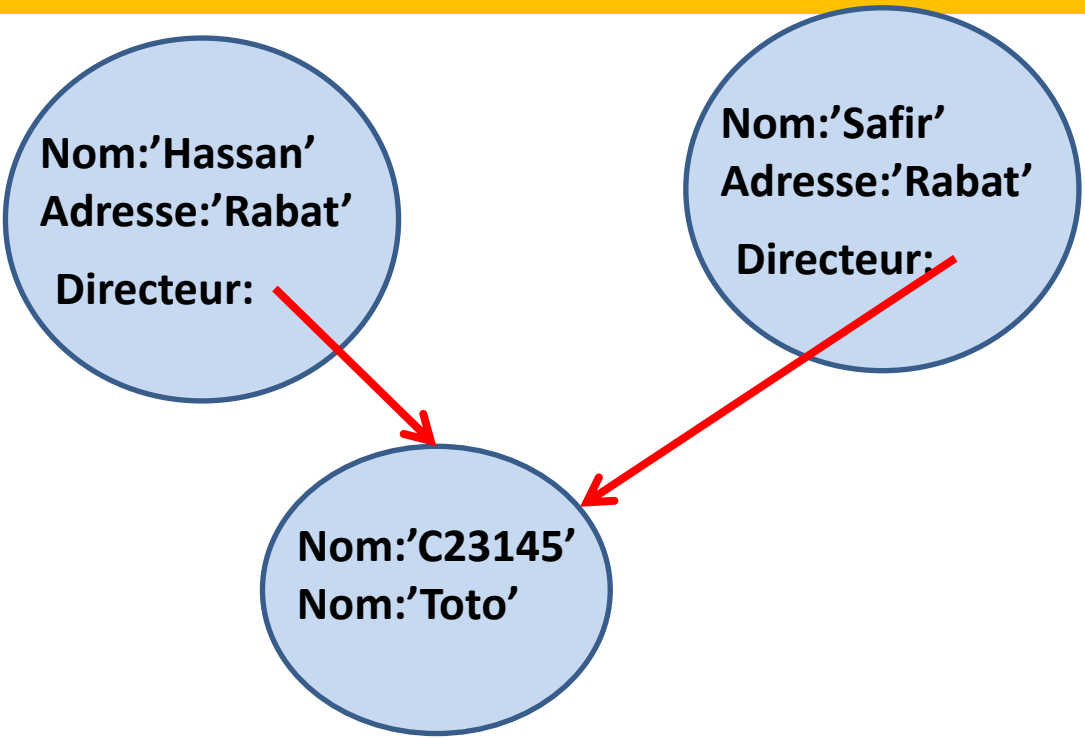
IDENTITE ET PARTAGE D'OBJET

```
class Personne{
private Integer    CIN;
private String     Nom;
.....
}
```

```
class Hotel {
private String     Nom;
private String     Adresse;
Private  Personne  Directeur;
.....
}
```

```
Personne pers=new Personne('C23145','toto');
Hotel HHass=new Hotel('Hassan','Rabat');
Hotel HSafir=new Hotel('Safir','Rabat');

HHass.Directeur=pers;
HSafir.Directeur=pers;
```



```
//ACCES PAR PLUSIEURS ENTREE
HHass.Directeur;
HSafir.Directeur;
```

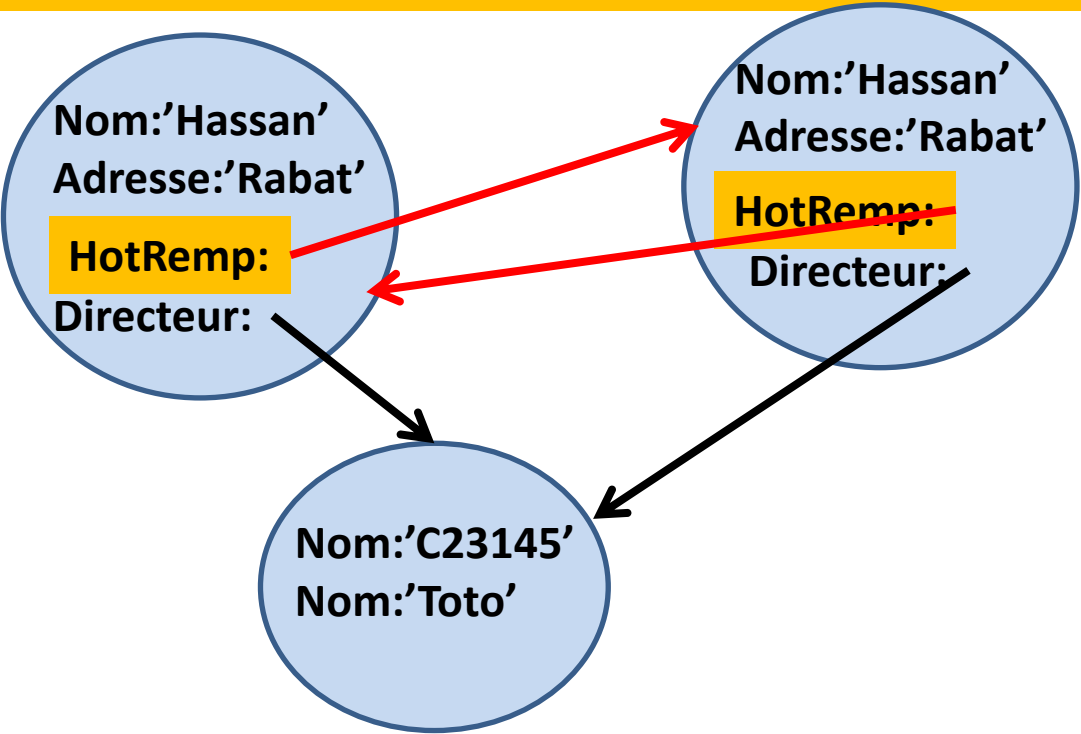
IDENTITE ET PARTAGE D'OBJET

```
class Personne{
private Integer    CIN;
private String    Nom;
.....
}
```

```
class Hotel {
private String    Nom;
private String    Adresse;
private Personne  Directeur;
private Hotel     HotRemp
.....
}
```

```
Personne pers=new Personne('C23145','toto');
Hotel HHass=new Hotel('Hassan','Rabat');
Hotel HSafir=new Hotel('Safir','Rabat');
```

```
HHass.Directeur=pers;
HSafir.Directeur=pers;
```



//ACCES PAR PLUSIEURS ENTREE

```
HHass.HotRemp=HSafir;
HSafir. HotRemp=HHass;
```

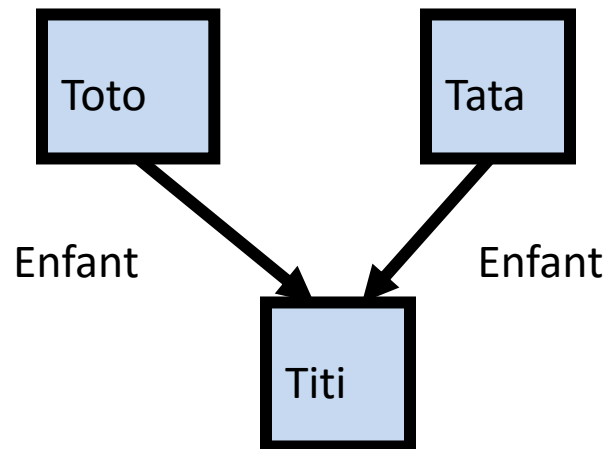
```
HHass.HotRemp.HotRemp ??
```

IDENTITE ET PARTAGE D'OBJET

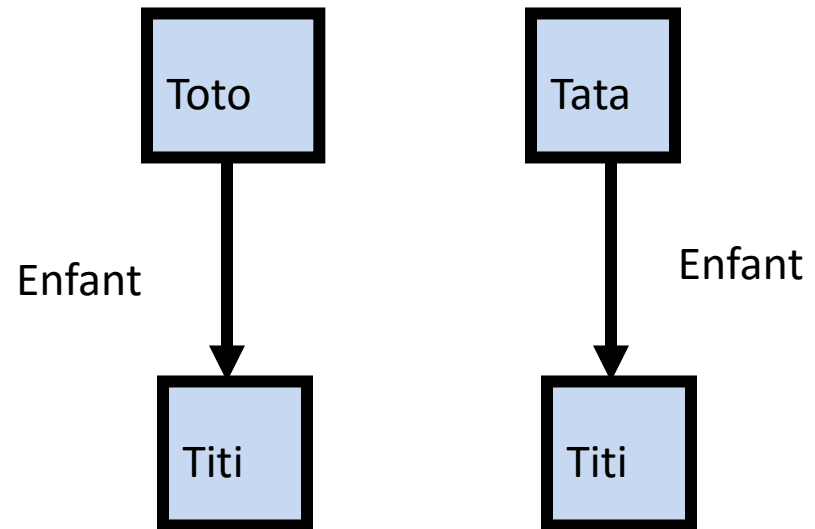
Possibilité de modéliser deux situations:

Toto a une fille nommée Titi

Tata a une fille nommée Titi



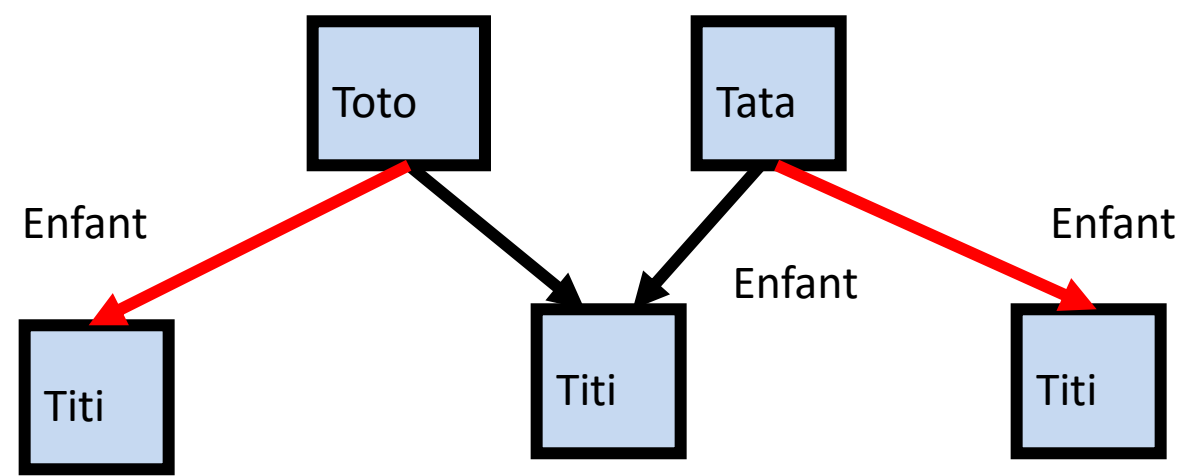
Situation 1: il s'agit de même enfant



Situation 2: il s'agit d'un enfant différent

IDENTITE ET PARTAGE D’OBJET

Possibilité de modéliser deux situations:
Toto a une fille nommée Titi d’un mariage antérieur
Tata a une fille nommée Titi d’un mariage antérieur
Toto et Tata ont une fille en commun qui s’appelle Titi



S’agit-il de même enfant: Toto→Enfant==tata→Enfant; ????????????????

IDENTITE ET PARTAGE D'OBJET: CONCLUSION

Représentation directe du monde réel

le SGBD voit les OBJETS (les OID) et le développeur voit les valeurs

OID et PRIMARY KEY sont deux notions différentes

On peut insérer plusieurs objets avec la même valeur. Pour le SGBD, il s'agit d'objets différents

Un objet n'est jamais copié ou dupliqué, il est référencé par son OID dans les autres objets

IDENTITE ET PARTAGE D'OBJET: CONCLUSION

Les Objets sont partagés à travers leurs OID

L'association entre les objets est implémentée à travers les OID

Il n'y a pas de contrainte d'intégrité référentielle et donc pas d'opérateur de jointure

Un nouveau mode de requête est possible: la navigation à travers le graphe des références.

DEFINITION D'UN SGBDOO

Aspect Orienté Objet

- Identité d 'objet
- **Objet complexe**
- Encapsulation
- Type ou Classe
- Héritage
- Surcharge et résolution tardive
- Complétude
- Extensibilité

- Les objets complexes sont construits à partir d 'objets atomiques et des constructeurs
- Les objets atomiques sont les entiers, les réels, les booléens, les chaînes de caractères.
- Les constructeurs sont le n-uplet (**tuple**), l 'ensemble (**set**), la liste (**list**) et le tableau (**array, vecteur**).
- L 'utilisation des constructeurs est indépendante du type des objets
- L 'utilisation des constructeurs peut être récursive

DEFINITION D'UN SGBDOO

Aspect Orienté Objet

- Identité d 'objet
- Objet complexe
- Encapsulation
- **Type ou Classe**
- Héritage
- Surcharge et résolution tardive
- **Complétude**
- **Extensibilité**

DEFINITION D'UN SGBDOO : NORME OMG

Les règles ouvertes

- **Modèle de calcul**
- **Modèle de persistance**
- **Uniformité**
- **Nommage**

DEFINITION D'UN SGBDOO : NORME OMG

Les règles ouvertes

- **Modèle de calcul**
- **Modèle de persistance**
- **Uniformité**
- **Nommage**

Rappel

- Dans les LPO: les objets sont Temporaires tant qu'ils ne sont pas écrits explicitement dans des fichiers
- Dans un SGBD
Les données persistent automatiquement

Qualités de la persistance

- Pour le même type on peut avoir des Instances persistantes ou pas
- Les opérations s'appliquent aux objets persistants ou temporaires
- Le statut persistant/temporaire peut être changé
- Un objet persistant ne peut référencer un objet temporaire

Techniques de persistance

- Nouvelles approches (SGBDOO)
 - Persistance à la demande
 - Persistance indépendante du type
- Différents modèle de persistance
 - Au niveau de l'objet ou de la Classe
 - Statique ou dynamique

Persistance de classe

- Une classe déclarée persistante alors toutes ses instances sont persistantes
- Les objets temporaires doivent être explicitement supprimés
- L'intégrité référentielle n'est pas Garantie

Exemple

Create persistent class Personne

Type tuple(nom : string,
 voiture : Véhicule)

Persistence de l'objet

- La sauvegarde est explicite par commande ou méthode
- L'intégrité référentielle n'est pas garantie

Exemple (O2)

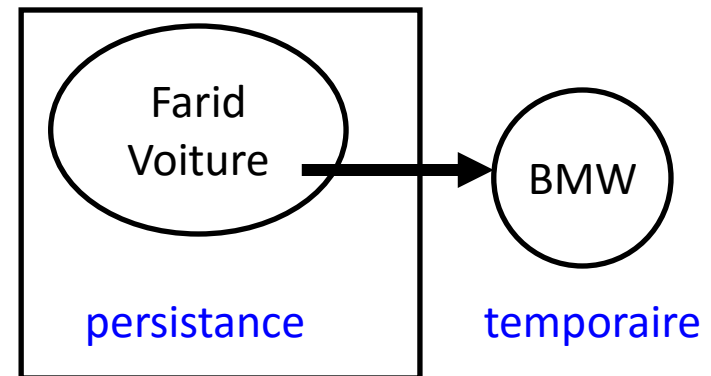
```
class Personne
Type tuple(          nom      : string,
                   voiture    : Véhicule)
```

```
Personne perso = new Personne;
Véhicule BMW = new Véhicule;
```

```
Perso->nom='Farid ;
```

```
Perso->voiture=BMW;
```

```
Put_object(perso);, //persistant
```



Persistence de l'objet: OMG

- L'objet est crée comme persistant ou temporaire
- Pas d'intégrité référentielle

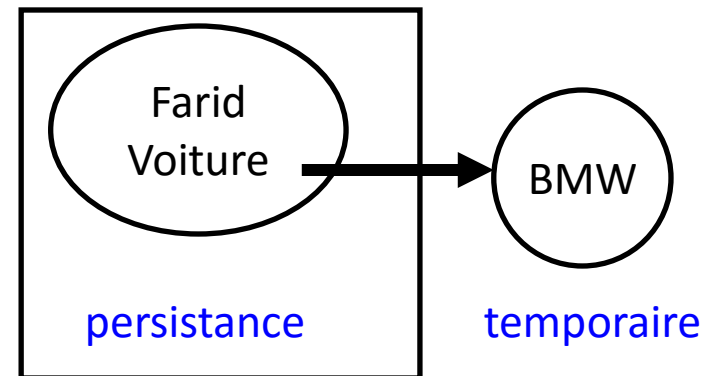
Exemple (OMG)

```
class Personne
Type tuple(          nom      : string,
                   voiture   : Véhicule)
```

```
Personne perso = new persistent Personne;
Véhicule BMW = new Véhicule;
```

```
Perso->nom='Farid ';
```

```
Perso->voiture=BMW;
```



Persistence par racine de persistance

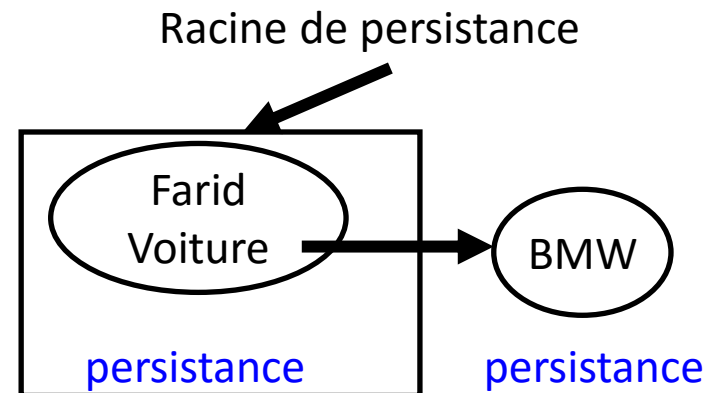
- Tout objet connecté directement ou par transitivité à une racine de persistance est persistant
- La racine de persistance est créée par nommage

Exemple (O2)

```
class Personne
Type tuple(          nom      : string,
                   voiture   : Véhicule)
Name les_persos=set(personne),
```

```
Personne perso = new Personne;
Véhicule BMW = new Véhicule;
Perso->nom='Farid';
Perso->voiture=BMW;
```

```
If ( newperso (perso->nom))
    les_persos+=set(perso)
Else ('erreur: existe déjà');
```



CONCLUSION

SGBDOO:

Capture aisée de la sémantique
Le développeur est très sollicité

SGBDR

Capture 'faiblement' la sémantique
Facilité de mise en œuvre
Le développeur n'est pas très sollicité

CONCLUSION

Vous connaissez les bases de données?

Vous connaissez la programmation objet?

Alors

Vous serez faire les bases de données objet!!!!

BD OO=programmation objet + persistance

EXEMPLE DE SGBDOO

VERSANT DB4O FOR JAVA

Objectifs

1. Découvrir un SGBDOO PUR OBJET
2. Donner les informations nécessaires pour faciliter l'écriture du TP sur db4o
3. Mini projet à comptabiliser dans la note finale

CARACTERISTIQUES

- Léger
- Pas de maintenance / administration ?
- Simple d'emploi
- API simple pour Java et .NET
- Requêtes natives (NQ), QBE, SODA ?
- Pas un SGBDOO ODMG,
- pas de requêtes OQL par exemple
- Client/Serveur ou local
- Convient bien à une application embarquée

APPORTS OBJET

- Identité objet
- Objet complexe
- Encapsulation
- Héritage
- Généricité
- Collections
- Transaction?

MODELE DE PERSISTANCE

- **PAR OBJET:** par stockage de l'objet dans la BD
- **PAR REFERENCE:** un objet référencé par un objet persistant devient automatiquement persistant
- **PAS DE RACINE DE PERSISTANCE:** mais possibilité de récupérer tous les objets d'une classe donnée dans la BD
- La base de données est un **CONTENAIR: ObjectContainer** associé à un fichier
- **Un seul fichier associé à la BD?**

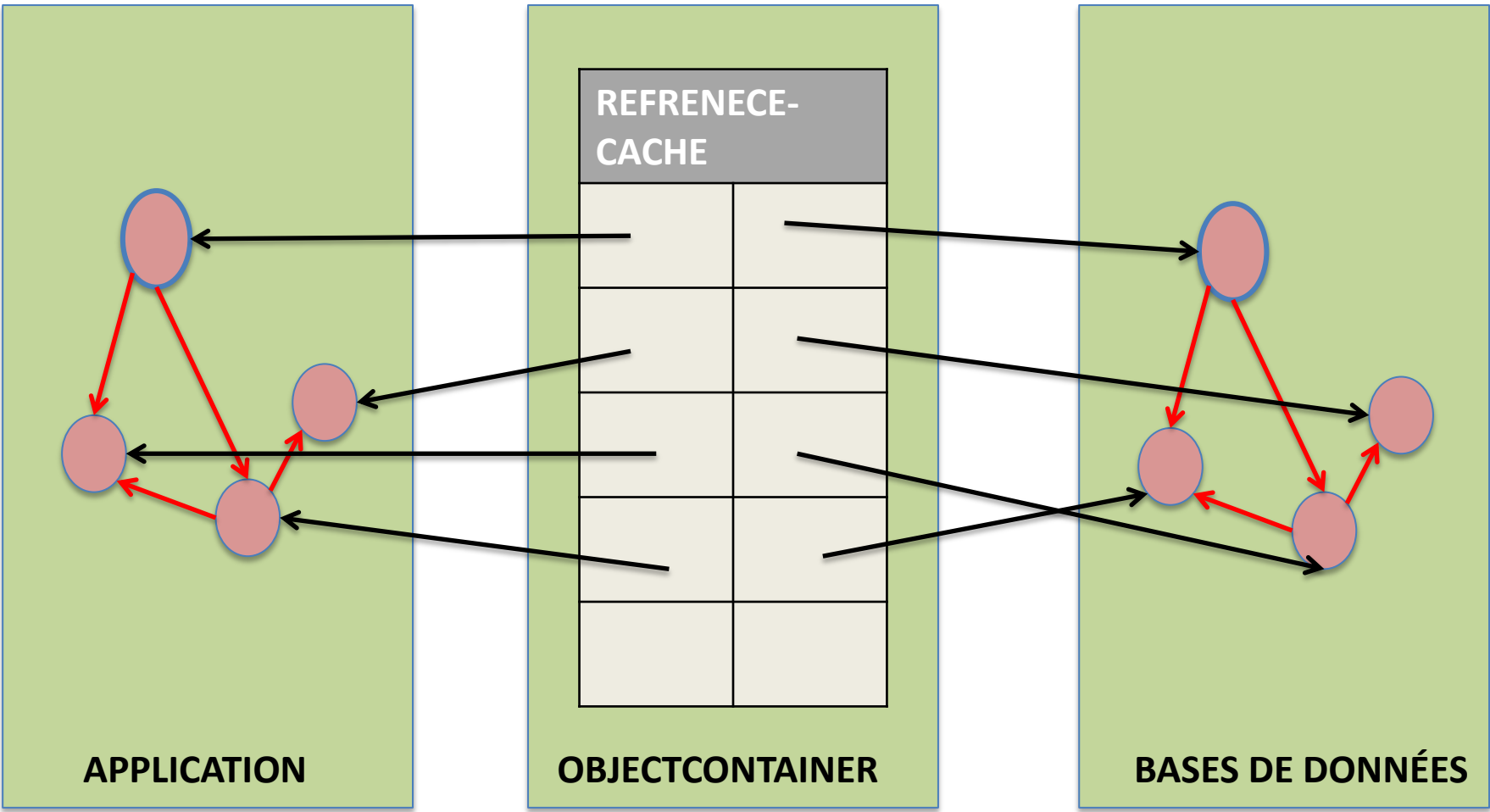
PAQUETAGES

Tous les paquets sont des sous paquets de `com.db4o`



Le paquetage `com.db4o` contient les interfaces et classes de base :

- `ObjectContainer`: contenir qui contient la BD
- `ObjectSet` : collection LIST pour les requetes
- `Db4oEmbedded` (la seule classe): moteur de DB4O



CREATION ET OUVERTURE DE LA BASE DE DONNEES

```
ObjectContainer db = Db4oEmbedded.openFile("databaseFile.db4o");

try {

    // utilisation des objets de la base

}
finally {
    db.close();
}
```

Remarques

1. Si la base n'existe pas, elle est créée sinon elle est ouverte
2. A la fermeture, les transactions sont automatiquement validées

STOCKAGE D'OBJET DANS LA BASE DE DONNEES

```
ObjectContainer bd = Db4oEmbedded.openFile("databaseFile.db4o");
try {
    Pilot pilot = new Pilot("toto");
    bd.store(pilot);
}
finally {
    bd.close();
}
```

Remarque

1. En mode locale, les transactions sont implicites
2. On peut valider quand même par db.commit()

RECUPERATION D'OBJET

```
ObjectContainer db = Db4oEmbedded.openFile("databaseFile.db4o");
try {
    List<Pilot> pilots = bd.queryByExample(new Pilot( "toto", null, null) );

    for (Pilot pilot : pilots) { System.out.println(pilot.getName()); }
}
finally {
    db.close();
}
```


RECUPERATION D'OBJET

```
ObjectContainer db = Db4oEmbedded.openFile("databaseFile.db4o");
try {
    List<Pilot> pilots = bd.query(new Predicate<Pilot>() {
        public boolean match(Pilot o) {
            return o.getName().equals(" toto");
        }
    });

    for (Pilot pilot : pilots) { System.out.println(pilot.getName()); }
}
finally {
    db.close();
}
```

MISE A JOUR (UPDATE) D'OBJET

```
ObjectContainer bd = Db4oEmbedded.openFile("databaseFile.db4o");
try {
    List<Pilot> pilots = bd.query(new Predicate<Pilot>() {
        public boolean match(Pilot o) {
            return o.getName().equals("toto");
        }
    });
    Pilot aPilot = pilots.get(0);
    aPilot.setName("titi");
    // mise à jour de l'objet
    bd.store(aPilot);
}
finally {
    bd.close();
}
```

SUPPRESSION (DELETE) D'OBJET

```
ObjectContainer bd = Db4oEmbedded.openFile("databaseFile.db4o");
try {
    List<Pilot> pilots = bd.query(new Predicate<Pilot>() {
        public boolean match(Pilot o) {
            return o.getName().equals("toto");
        }
    });
    Pilot aPilot = pilots.get(0);
    bd.delete(aPilot);
}
finally {
    bd.close();
}
```

DB4O

EXEMPLE

```
package tp1;
public class Personne {
    private int age;
    private String nom;
    private String prenom;

    public Personne(String nom, String prenom, int age) {
        super();
        this.age = age;
        this.nom = nom;
        this.prenom = prenom;
    }

    public int getAge() {return age;}
    public void setAge(int age) {this.age = age; }
    public String getNom() {return nom; }
    public void setNom(String nom) {this.nom = nom; }
    public String getPrenom() {return prenom; }
    public void setPrenom(String prenom) {this.prenom = prenom; }

    // la fonction toString
    @Override
    public String toString() {
        return "Personne [ nom=" + nom + ", prenom=" + prenom + ", age=" + age + " ]";
    }
}
```

```
package tp1;
import com.db4o.Db4oEmbedded;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;

public class Main {
    public static void main(String[] args) {
        ObjectContainer bd = null;
        bd = Db4oEmbedded.openFile("bd.data");
        Personne personne100 = new Personne("toto", "titi", 35);
        .....
        // Ajoute les personnes dans la base de données
        bd.store(personne100);
        bd.commit();

        //bd.commit();
        // Passe un objet, exemple de ce que l'on cherche.

        ObjectSet<Personne> personnes = bd.queryByExample(new Personne(null,null, 40));
        for (Personne personne : personnes) {
            System.out.println(personne);
        }
        bd.close();
    }
}
```

DB4O

MODE DE REQUETES QBE, SODA et NATIVE

QBE

QBE : EXEMPLE 1

// TOUS LES OBJETS DE TYPE PERSON

```
System.out.println("EXEMPLE 1: TOUS LES OBJETS DE TYPE PERSON");  
res = db.queryByExample(new Person());  
listResult(res);
```

EXEMPLE 1: TOUS LES OBJETS DE TYPE PERSON

[Kadour;79]

[Zaatar;56]

[Ayachi;86]

[Dahman;86]

[Bouchaib;42]

[Khalfi;82]

QBE : EXEMPLE 2

// RECHERCHE PAR NOM

```
System.out.println("EXEMPLE 2: RECHERCHE PAR NOM");  
template = new Person();  
template.setName("Bouchaib");  
res = db.queryByExample(template);  
listResult(res);
```

EXEMPLE 2: RECHERCHE PAR NOM
[Bouchaib;42]

QBE : EXEMPLE 3

// RECHERCHE PAR NOM ET PAR AGE

```
System.out.println("EXEMPLE 3: RECHERCHE PAR NOM ET PAR AGE");  
template = new Person();  
template.setName("Bouchaib");  
template.setAge(42);  
res = db.queryByExample(template);  
listResult(res);
```

EXEMPLE 3: RECHERCHE PAR NOM ET PAR AGE**[Bouchaib;42]**

QBE : EXEMPLE 4

// RECHERCHE PAR AGE

```
System.out.println("EXEMPLE 4: RECHERCHE PAR AGE");  
template = new Person(null,82);  
res = db.queryByExample(template);  
listResult(res);
```

EXEMPLE 4: RECHERCHE PAR AGE
[Khalfi;82]

QBE : EXEMPLE 5

```
// RECHERCHE DE TOUS LES OBJETS DE LA CLASSE PERSON
System.out.println("EXEMPLE 5: RECHERCHE DE TOUS LES OBJETS EN UTILISANT LA
CLASSE");
res = db.queryByExample(Person.class);
listResult(res);
```

EXEMPLE 5: RECHERCHE DE TOUS LES OBJETS EN UTILISANT LA CLASSE

```
[Kadour;79]
[Zaatar;56]
[Ayachi;86]
[Dahman;86]
[Bouchaib;42]
[Khalfi;82]
```

QBE : EXEMPLE 6

```
// RECHERCHE DE TOUS LES OBJETS EN UTILISANT NULL
System.out.println("EXEMPLE 6: TOUS LES OBJETS");
res = db.queryByExample(null);
listResult(res);
```

EXEMPLE 5: RECHERCHE DE TOUS LES OBJETS EN UTILISANT LA CLASSE

```
[Kadour;79]
[Zaatar;56]
[Ayachi;86]
[Dahman;86]
[Bouchaib;42]
[Khalfi;82]
```

REQUETES SODA

La classe QUERY: méthodes et opérateurs

Query : INTERFACE REQUETE

constrain:	AJOUTE UNE CONTRAINTE AU NOEUD COURANT
descend:	DESCEND VERS LE NOEUD FILS (CHAMPS)
orderAscending:	ORDRE ASCNEDANT: CONTRAINTE SUR LE NOEUD COURANT
orderDescending:	ORDRE DESCENDANT: CONTRAINTE SUR LE NOEUD COURANT
execute:	EXECUTE LA REQUETE ET RETOURNE LA LISTE DES OBJETS

LES OPERATEURS

and
or
equal
greater
smaller

SODA : EXEMPLE 12**// TOUS LES PERSONNES**

```
System.out.println("EXEMPLE 12: SODA - TOUS LES PERSONS");  
query = db.query();  
query.constrain(Person.class);  
res2 = query.execute();  
listResult(res2);
```

EXEMPLE 12: SODA - TOUS LES PERSONS**[Kadour;79]****[Zaatar;56]****[Ayachi;86]****[Dahman;86]****[Bouchaib;42]****[Khalfi;82]**

SODA : EXEMPLE 13**// PAR NOM**

```
System.out.println("EXEMPLE 13: SODA - PAR NOM");  
query = db.query();  
query.constrain(Person.class);  
query.descend("_name").constrain("Zaatar");  
res2 = query.execute();  
listResult(res2);
```

EXEMPLE 13: SODA - PAR NOM**[Zaatar;56]**

SODA : EXEMPLE 14**// AVEC L'OPERATEUR NOT**

```
System.out.println("EXEMPLE 14: SODA - AGE NOT 56");  
query = db.query();  
query.constrain(Person.class);  
query.descend("_age").constrain(56).not(); // not 56  
res2 = query.execute();  
listResult(res2);
```

EXEMPLE 14: SODA - AGE NOT 56**[Kadour;79]****[Ayachi;86]****[Dahman;86]****[Bouchaib;42]****[Khalfi;82]**

SODA : EXEMPLE 15

// Composition de query

```
System.out.println("EXEMPLE 15: SODA - AGE= 86 AND NAME = 'Ayachi'");  
query = db.query();  
query.constrain(Person.class);  
Constraint firstConstr = query.descend("_age").constrain(86); // 1er constraint  
query.descend("_name").constrain("Ayachi").and(firstConstr); // éime utilisant and  
res2 = query.execute();  
listResult(res2);
```

EXEMPLE 15: SODA - AGE= 86 AND NAME = 'Ayachi'
[Ayachi;86]

SODA : EXEMPLE 16**// AND, AUTRE VERSION**

```
System.out.println("EXEMPLE 16: SODA - AGE= 86 AND NAME = 'Ayachi");  
query = db.query();  
query.constrain(Person.class);  
query.descend("_age").constrain(86); // first constraint  
query.descend("_name").constrain("Ayachi");  
res2 = query.execute();  
listResult(res2);
```

EXEMPLE 16: SODA - AGE= 86 AND NAME = 'Ayachi' (ALT)
[Ayachi;86]

SODA : EXEMPLE 17

```
// "Or" query
System.out.println("EXEMPLE 17: SODA - AGE= 86 OR NAME = 'DAHMAN'");
query = db.query();
query.constrain(Person.class);
firstConstr = query.descend("_age").constrain(86); // 1ere constraint
query.descend("_name").constrain("Dahman").or(firstConstr); // 2me utilisant OR
res2 = query.execute();
listResult(res2);
```

EXEMPLE 17: SODA - AGE= 86 OR NAME = 'DAHMAN'

[Ayachi;86]

[Dahman;86]

SODA : EXEMPLE 18**// INTERVAL query**

```
System.out.println("EXEMPLE 18: SODA - AGE BETWEEN 60 AND 80");  
query = db.query();  
query.constrain(Person.class);  
firstConstr = query.descend("_age").constrain(60).greater(); // 1er constraint  
query.descend("_age").constrain(80).smaller().and(firstConstr); // 2me utilisant and  
res2 = query.execute();  
listResult(res2);
```

EXEMPLE 18: SODA - AGE BETWEEN 60 AND 80
[Kadour;79]

SODA : EXEMPLE 19

```
// "Greater" query
System.out.println("EXEMPLE 19: SODA - AGE > 80");
query = db.query();
query.constrain(Person.class);
query.descend("_age").constrain(80).greater();
res2 = query.execute();
listResult(res2);
```

EXEMPLE 19: SODA - AGE > 80

[Ayachi;86]

[Dahman;86]

[Khalfi;82]

SODA : EXEMPLE 20**// "Like" query**

```
System.out.println("EXEMPLE 20: SODA - LIKE 'Ka'");  
query=db.query();  
query.constrain(Person.class);  
query.descend("_name").constrain("Ka").like();  
res2 = query.execute();  
listResult(res2);
```

EXEMPLE 20: SODA - LIKE 'Ka'
[Kadour;79]

SODA : EXEMPLE 21**// TRI query**

```
System.out.println("EXEMPLE 21:TRI SODA PAR NOM - TOUS LES PERSONS");  
query = db.query();  
query.constrain(Person.class);  
query.descend("_name").orderAscending();  
res2 = query.execute();  
listResult(res2);
```

EXEMPLE 21:TRI SODA PAR NOM - TOUS LES PERSONS

```
[Ayachi;86]  
[Bouchaib;42]  
[Dahman;86]  
[Kadour;79]  
[Khalfi;82]  
[Zaatar;56]
```

SODA : EXEMPLE 22

```
// Null/zero value query
// POUR LES VALEURS NON DEFINIES DANS LES CHAMPS DE L'OBJET
Person p11 = new Person();
p11.setName("Toto");
db.store(p11);
db.commit();
System.out.println("EXEMPLE 22:SODA - NULL/ZERO VALUE");
query=db.query();
query.constrain(Person.class);
query.descend("_age").constrain(0);
res2=query.execute();
listResult(res2);
```

EXEMPLE 22:SODA - NULL/ZERO VALUE
[Toto;0]

REQUETES NATIVES (NQ)

LA CLASSE PERSONNE

```
package chap6query;

public class Person implements Comparable {
    private String _name;
    private int _age;

    public Person(){}

    public Person(String name, int age) {
        _name = name;
        _age = age;
    }

    public int getAge() {
        return _age;
    }

    public void setAge(int value) {
        _age = value;
    }
}
```

```
public String getName() {
    return _name;
}

public void setName(String value) {
    _name = value;
}

@Override
public String toString() {
    return "[" + _name + ";" + _age + "]";
}

@Override
public int compareTo(Object o) throws
ClassCastException {
    Person p = (Person) o;
    return
this.getName().compareTo(p.getName());
}
}
```

LA CLASSE PersonPredicate

```
package chap6query;

import com.db4o.query.*;

//cette classe implémente Predicate fourni dans java.util.function.Predicate
//il permet de définir un pattern. Si l'objet correspond au pattern, on retourne true
//sinon on retourne false
//dans java 8, c'est une classe Interface Fonctionnelle pour passer des fonction en
//argument d'une autre fonction

public class PersonPredicate extends Predicate{

    @Override
    public boolean match(Object o) {
        return true;
    }

}
```

CREATION, OUVERTURE DE LA BASE DE DONNEES ET INSTANCIATION

```
ObjectContainer db = Db4oEmbedded.openFile("queries.db4o");  
    Person p1=new Person("Kadour", 79);  
    Person p2=new Person("Zaatar", 56);  
    Person p3=new Person("Ayachi", 86);  
    Person p4=new Person("Dahman", 86);  
    Person p5=new Person("Bouchaib", 42);  
    Person p6=new Person("Khalfi", 82);
```

```
//stockage dans la BD
```

```
    db.store(p1);  
    db.store(p2);  
    db.store(p3);  
    db.store(p4);  
    db.store(p5);  
    db.store(p6);
```

```
private static void listResult(ObjectSet res){  
    while(res.hasNext())  
        System.out.println(res.next());  
}
```

REQUETES NATIVES : PRINCIPE

La classe doit spécialiser Predicate

Elle doit implémenter la méthode match()

public boolean match(argument)

La méthode peut déclarer des variables locales

Le constructeur ou les méthodes set() de

la classe permettent le passage de critères de sélection

NQ: SYNTAXE 1

```
public <TargetType extends Object> ObjectSet<TargetType> query(  
Predicate<TargetType> prdct  
)
```

NQ : EXEMPLE 23

```
// RECHERCHE DES PERSONNES DONT L'AGE EST > 60
System.out.println("EXEMPLE 7: SIMPLE NQ - AGE > 60");
persons = db.query ( new Predicate<Person>() {
    @Override
    public boolean match(Person person) {
        return person.getAge() > 60;
    }
} );
for (Person person : persons)
    System.out.println(person);
```

EXEMPLE 23:TRI SODA PAR NOM - TOUS LES PERSONS

```
[Ayachi;86]
[Dahman;86]
[Kadour;79]
[Khalfi;82]
```

NQ : EXEMPLE 24

// AVEC L'OPERATEUR OU

```
System.out.println("EXEMPLE 8: RANGE NQ - AGE BETWEEN 60 AND 80");
persons = db.query(new Predicate<Person>() {
    @Override
    public boolean match(Person person) {
        return person.getAge() < 60 || person.getAge() > 80;
    }
});
for (Person person : persons)
    System.out.println(person);
```

EXEMPLE 24: RANGE NQ - AGE BETWEEN 60 AND 80

[Zaatar;56]

[Ayachi;86]

[Dahman;86]

[Bouchaib;42]

[Khalfi;82]

NQ : EXEMPLE 25

```
// AVEC L'OPERATEUR AND
```

```
System.out.println("EXEMPLE 9: NQ - AGE >80 AND NAME='KHALFI'");
persons = db.query(new Predicate<Person>() {
    @Override
    public boolean match(Person person) {
        return person.getAge() > 80 && person.getName().equals("Khalfi");
    }
});
for (Person person : persons)
    System.out.println(person);
```

EXEMPLE 25: NQ - AGE >80 AND NAME='KHALFI'
[Khalfi;82]

NQ : EXEMPLE 26: TRI DES RESULTATS AVEC UN COMPAREUR

// Query avec Comparator
//SYNTAXE

```
public <TargetType extends Object> ObjectSet<TargetType> query(  
    Predicate<TargetType> prdct,  
    Comparator<TargetType> cmptr  
)  
throws Db4oIOException, DatabaseClosedException;
```

NQ : EXEMPLE 26

```
// Query avec Comparator  
//SYNTAXE
```

```
public <TargetType extends Object> ObjectSet<TargetType> query(  
    Predicate<TargetType> prdct,  
    Comparator<TargetType> cmptr  
)  
throws Db4oIOException, DatabaseClosedException;
```

```
// UTILISATION DU Comparator  
// REDEFINITION DU COMPARATOR  
Comparator<Person> personCmp=new Comparator<Person>() {  
    @Override  
    public int compare(Person o1, Person o2) {  
        return o1.getName().compareTo(o2.getName());  
    }  
};
```

NQ : EXEMPLE 26

```
ObjectSet<Person> result = db.query(new Predicate<Person>() {  
    public boolean match(Person person) {  
        return true;  
    }  
}, personCmp);  
  
for (Person person : result)  
    System.out.println(person);
```

EXEMPLE 26: TRI NQ - TOUS LES PERSONS

```
[Ayachi;86]  
[Bouchaib;42]  
[Dahman;86]  
[Kadour;79]  
[Khalfi;82]  
[Zaatar;56]
```

NQ : EXEMPLE 27: TRI DES RESULTATS EN UTILISANT UN TABLEAU

// TRI UTILISANT UN TABLEAU

```
System.out.println("EXEMPLE 11: TRI NQ - UTILISANT UN TABLEAU ARRAY");
persons = db.query(new Predicate<Person>() {
    public boolean match(Person person) {
        return true;
    }
});
Object p[] = persons.toArray();
System.out.println("avant le tri");
Arrays.sort(p);
for(int i=0; i<p.length; i++) {
    System.out.println(p[i]);
}
```

EXEMPLE 27: TRI NQ - UTILISANT UN TABLEAU ARRAY

avant le tri

[Ayachi;86]

[Bouchaib;42]

[Dahman;86]

[Kadour;79]

[Khalfi;82]

[Zaatar;56]

ASPECTS AVANCES: PERSISTANCE ET OBJETS COMPLEXES

LA CLASSE PERSONNE

```
package tp4_activation;

public class Personne {
    private String prenom;
    private Personne mere;

    public Personne(String prenom, Personne
mere) {
        this.prenom = prenom;
        this.mere = mere;
    }

    public Personne(String prenom) {
        this.prenom = prenom;
    }

    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
}
```

```
public void setMere(Personne mere) {
    this.mere = mere;
}

public String getPrenom() {
    return prenom;
}

public Personne getMere() {
    return mere;
}

@Override
public String toString() {
    return "Personne{" + "prenom=" +
prenom + '}';
}
}
```

AFFICHAGE

```
ObjectContainer db=Db4oEmbedded.openFile("ACTIVATION_BDTEST.DB4O");
```

```
Personne Zahra=new Personne("Zahra");  
Personne Fatouma=new Personne("Fatouma");  
Personne Khadouj=new Personne("Khadouj");  
Personne Rahma=new Personne("Rahma");  
Personne Fetouche=new Personne("Fetouche");  
Personne Daouya=new Personne("Daouya");
```

```
//les filations  
Zahra.setMere(Fatouma);  
Fatouma.setMere(Khadouj);  
Khadouj.setMere(Rahma);  
Rahma.setMere(Fetouche);  
Fetouche.setMere(Daouya);
```

```
//Stockage dans la BD  
db.store(Zahra);
```

ASPECTS AVANCES: ACCES OBJETS COMPLEXES

AFFICHAGE

```
//recherche
ObjectSet res=db.queryByExample(new Personne("Zahra", null));

Personne Zah=(Personne)res.get(0);

System.out.println("AFFICHAGE DE L AILLEULE DE ZAHRA");
Personne ailleule = Zah.getMere().getMere().getMere().getMere().getMere();
System.out.println(ailleule.getPrenom());
```

```
run:
AFFICHAGE DE L AILLEULE DE ZAHRA
null
BUILD SUCCESSFUL (total time: 1 second)
```

MODIFICATION

```
//recherche et test
ObjectSet res=db.queryByExample(new
Personne("Zahra", null));
Personne Zah=(Personne)res.get(0);

//modification
System.out.println("MODIFICATION DU
ZAHRA A ZAH RATOUN");
System.out.println("ET MODIFICATION DE SA
MERE Maman de Zahra");
Zah.setPrenom("Zahratoun");
Zah.getMere().setPrenom("Maman de
Zahra");
db.store(Zah);
db.commit();
```

```
//verification
System.out.println("VERIFICATION DE
ZAHRA ET DE SA MERE");
System.out.println("TOUJOURS DANS
LA MEME SESSION");
ObjectSet
resAM=db.queryByExample(new
Personne("Zahratoun", null));
Personne
ZahAM=(Personne)resAM.get(0);
Personne MereAM=ZahAM.getMere();
```

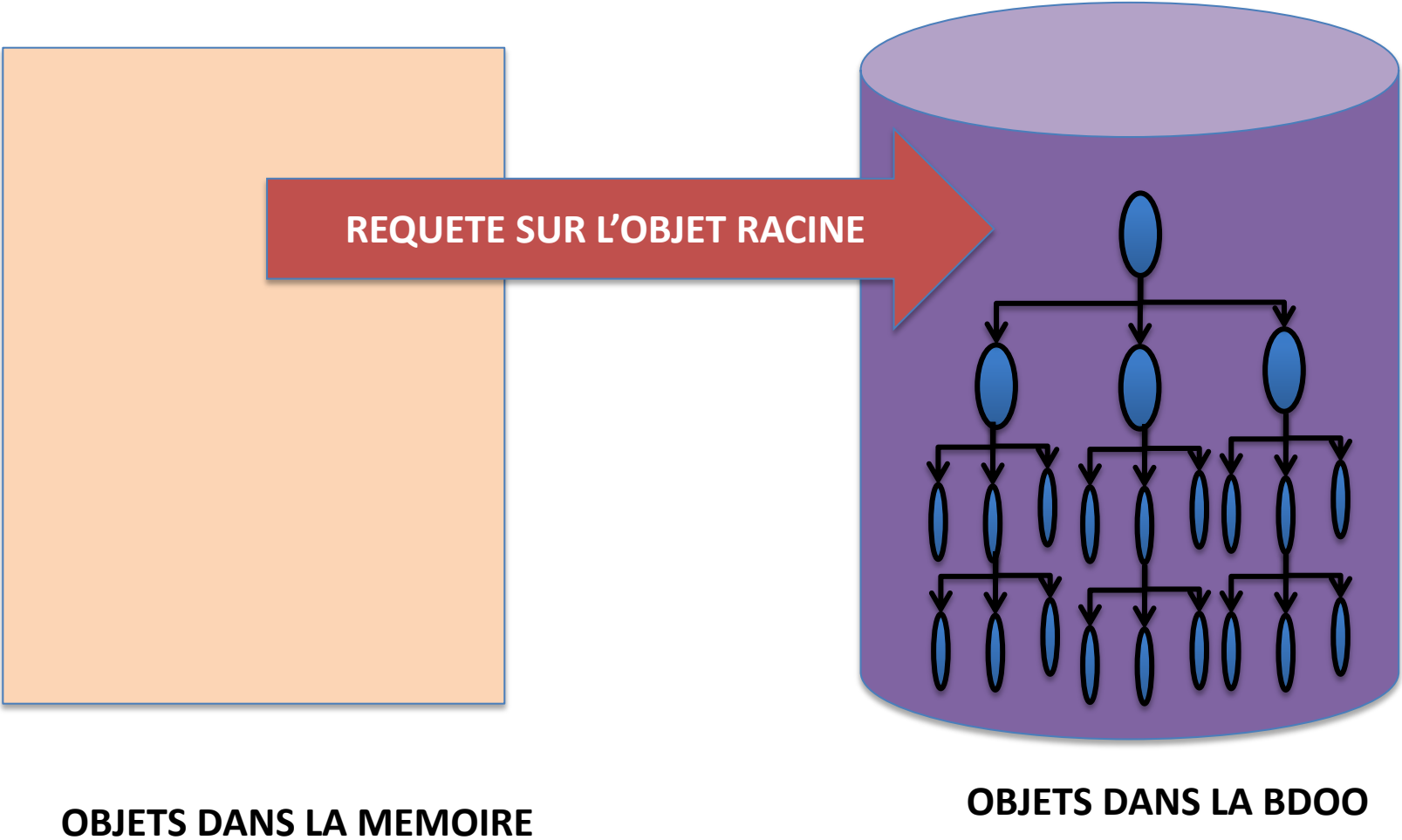
```
run:
MODIFICATION DU ZAHRA A ZAH RATOUN
ET MODIFICATION DE SA MERE Maman de
Zahra
VERIFICATION DE ZAHRA ET DE SA MERE
TOUJOURS DANS LA MEME SESSION
Zahratoun
Maman de Zahra
```

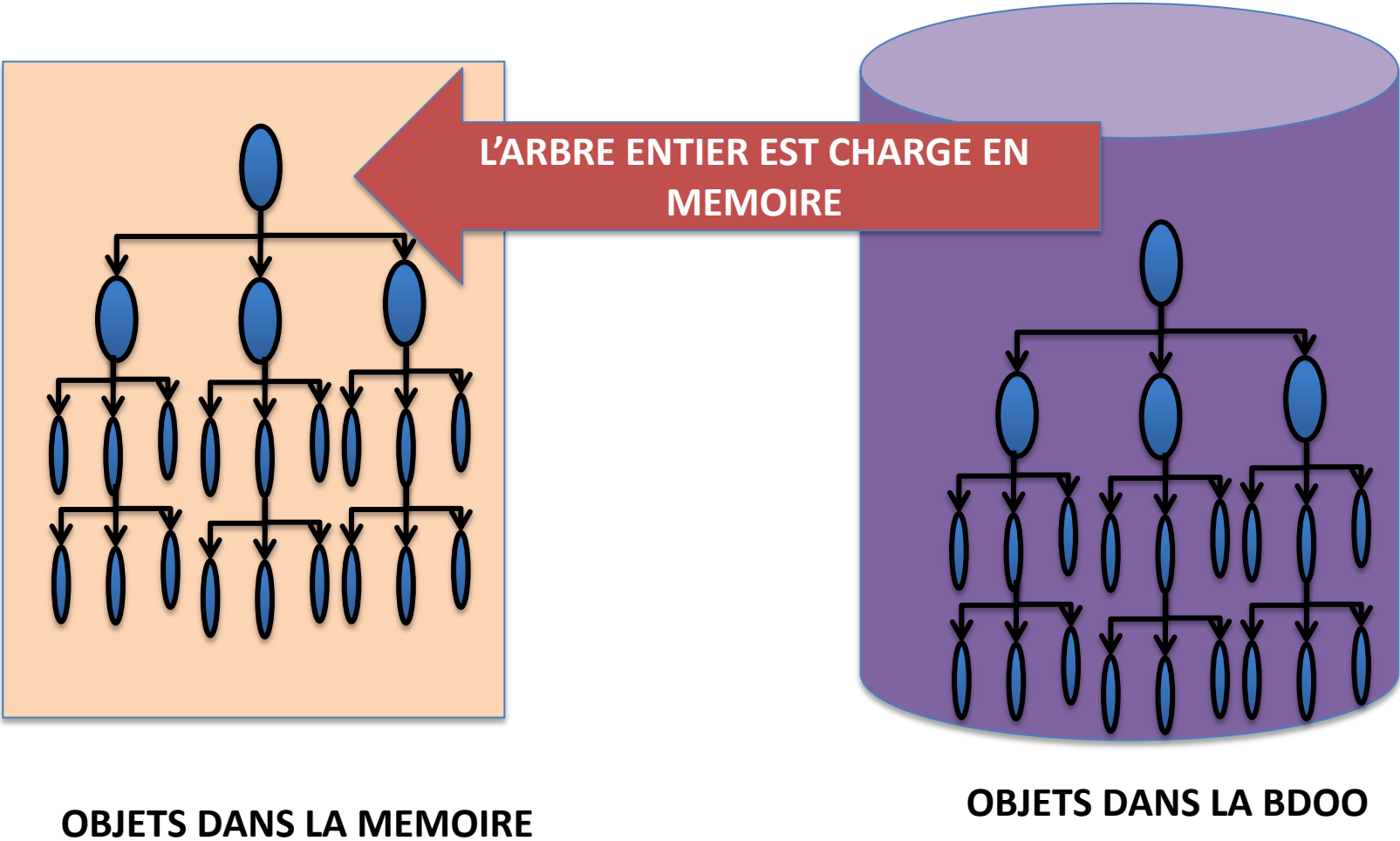
MODIFICATION

```
//fermeture de la BD
    db.close();
    ObjectContainer
dbAMF=Db4oEmbedded.openFile("ACTIVATION
_BDTEST.DB4O");
    //verification
    System.out.println("VERIFICATION DE
ZAHRA ET DE SA MERE");
    System.out.println("APRES LA FERMETURE
ET LA REOUVERTURE DE LA BD");
    ObjectSet
resAMF=dbAMF.queryByExample(new
Personne("Zahratoun", null));
    Personne
ZahAMF=(Personne)resAMF.get(0);
    Personne MereAMF=ZahAMF.getMere();

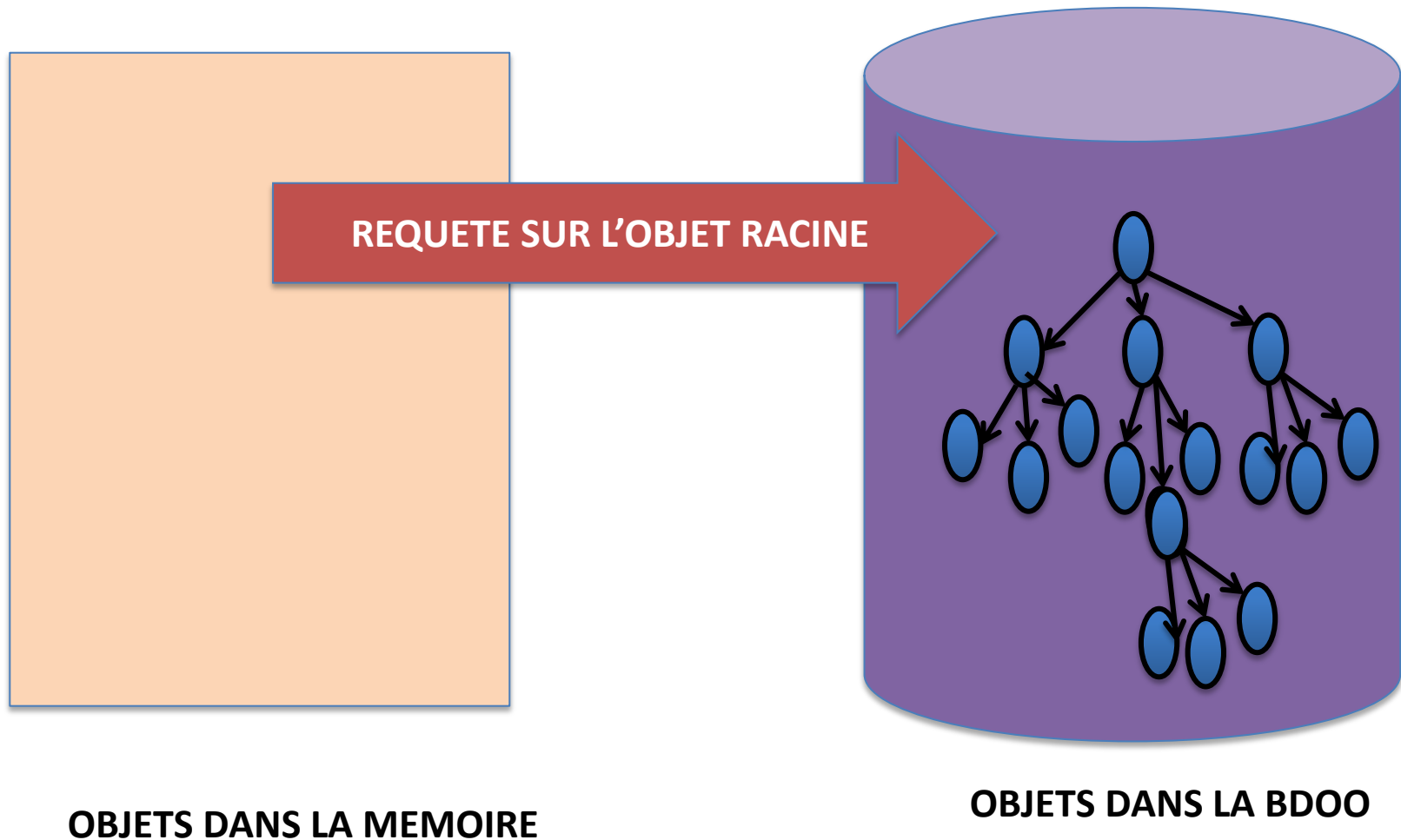
System.out.println(MereAMF.getPrenom());
    dbAMF.close();
```

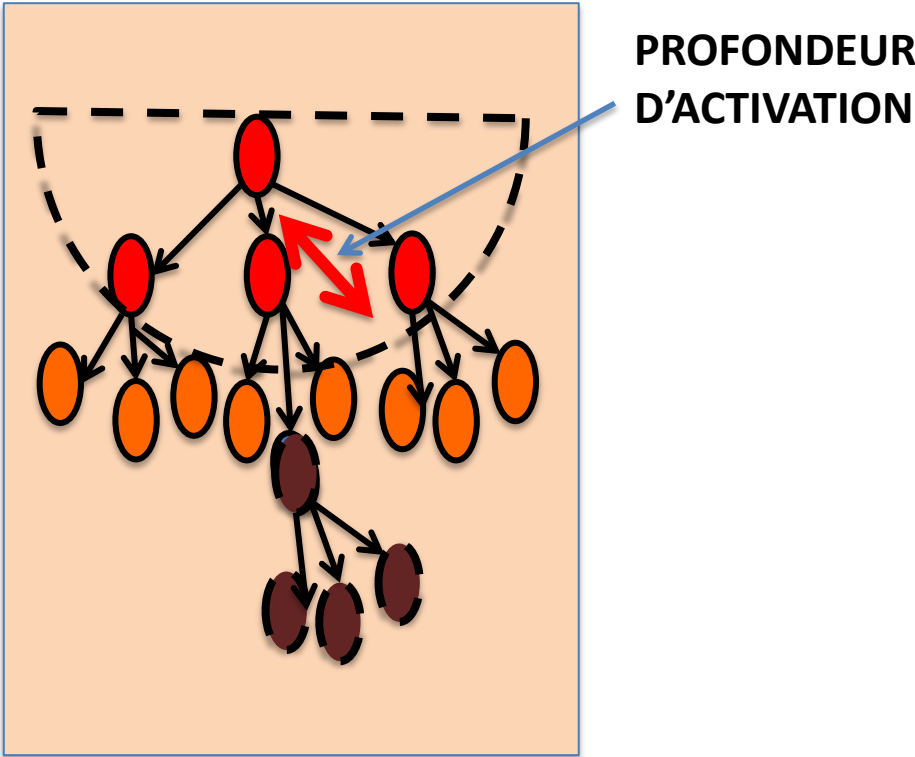
```
VERIFICATION DE ZAHRA ET DE SA MERE
APRES LA FERMETURE ET LA REOUVERTURE
DE LA BD
Zahratoun
Fatouma
BUILD SUCCESSFUL (total time: 0 seconds)
```



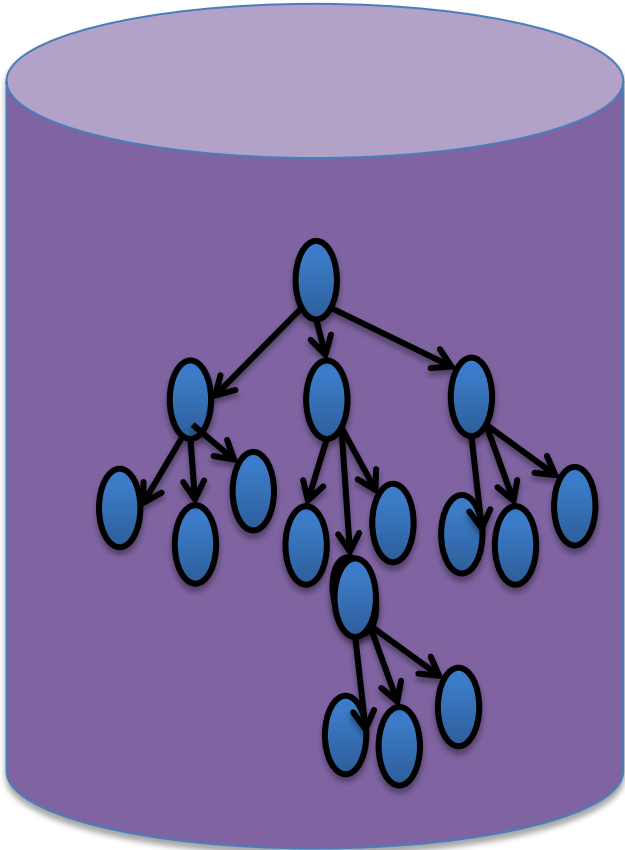


PROBLEME !!!!! SOLUTION ??????





OBJETS DANS LA MEMOIRE



OBJETS DANS LA BDOO

-  OBJET ACTIF
-  OBJET INACTIF
-  OBJET NON CHARGE

REGLES D'ACTIVATIONS DANS DB4O

1. QUAND UN OBJET EST ITÉRÉ, IL EST ACTIVÉ AUTOMATIQUEMENT
2. ACTIVE EXPLICITEMENT PAR LA METHODE ACTIVATE DU CONTENAIRE
`db.activate(ailleule,5);`
3. LES ELEMENTS DE LA COLLECTION SONT ACTIVES AUTOMATIQUEMENT POUR:
UNE PROFONDEUR 1 POUR LES LISTES
UNE PROFONDEUR 2 POUR LES MAP

```
import com.db4o.config.EmbeddedConfiguration;
```

```
EmbeddedConfiguration config = Db4oEmbedded.newConfiguration();  
  
config.common().activationDepth(6);  
  
config.common().objectClass(Personne.class).cascadeOnActivate(true);  
  
objectClass(Personne.class).cascadeOnDelete(true);  
  
ObjectContainer db=Db4oEmbedded.openFile(config,"BD_1_ACT.DB4O");
```

LE MODELE RELATIONNEL ETENDU (LE RELATIONNEL OBJET/SQL3)

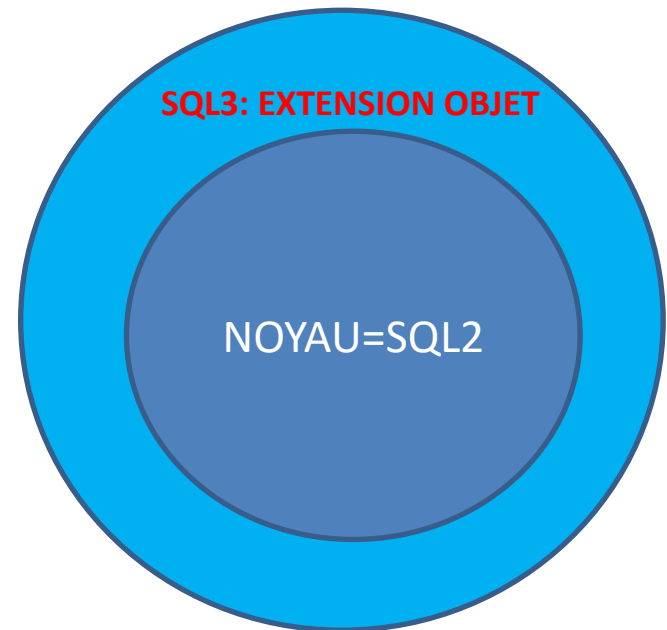
SQL3 = extension du SQL aux concepts objets

Le noyau est toujours relationnel

La table reste le support de prédilection pour la manipulation des données

Les SGBDs basés sur SQL3 sont dits:
OBJET-RELATIONNELS
ou
RELATIONNELS ETENDUS

Exemple: Oracle, Informix, Sybase, DB2...



Les objets d'un type ne deviennent persistants que lorsqu'ils sont insérés dans une table créée avec CREATE TABLE

Exemple d'objet complexe possible dans SQL3

Police	Nom	Adresse	Conducteurs		Accidents		
24	Toto	Casa	Conducteur	Age	Accident	Rapport	Photo
			Tata	45	134		
			Titl	17			
					219		
					037		

Exemple d'objet complexe d'Oracle

Class Assurance

Type (

Police

Integer,

Nom

String,

Adresse String,

Conducteurs

List(C_Conducteurs),

Accidents

List(C_Accidents))

.....

End;

Class C_conducteurs

Type (

Conducteur

String,

Age

Integer,

)

.....

End;

Exemple: implémentation de la table en BDOO pure

NON PREMIERE FORME NORMALE

Forme normale tolérant des domaines multivalués

MODELE RELATIONNEL IMBRIQUE

Un domaine peut lui même être valué par des tables de même schéma

L'APPORT DU MODELE OBJET

Objets complexes et type utilisateur

identité d'objet (REF)

Encapsulation des données

Héritage d'opérations et de structures

Collections (varray, nested table)

Analogie BDOO/R-OO

SGBDOO	Relationnel étendu (ROO)	Commentaire
OBJET	ADT	-pas de véritable objet -pas d'encapsulation
IDENTITE d'OBJET	(REFERENCE) POINTEUR	-les références sont gérées par le développeur explicitement
COLLECTION	VARRAY NESTED TABLE	-lourd à gérer -des tables référencées par la table mère
HERITAGE	HERITAGE DE TYPE HERITAGE DE TABLE	COMPLIQUE

LE MODELE RELATIONNEL ETENDU VUE GENERALE DE L'APPORT DE L'OBJET

TYPE ABSTRAIT

LE MODELE RELATIONNEL ETENDU TYPE ABSTRAIT et REFERENCE

Les types abstraits (ADT)

SPECIFICATION DE TYPE

DECLARATION DES ATTRIBUTS

DECLARATION DES METHODES

CORPS DU TYPE

IMPLEMENTATION DES METHODES

Les types abstraits: exemple

SQL>

SQL> create or replace **type** tadr **as object**(

2 numero number(3),

3 rue varchar2(20),

4 code_postal number(5),

5 ville varchar2(20)

6);

7 /

Type créé.

SQL>

Les types abstraits: exemple

```

• CREATE TYPE t_adresse AS OBJECT(
•     Numero          NUMBER(4),
•     Rue              VARCHAR2(20),
•     Code_postal      NUMBER(5),
•     Ville            VARCHAR2(20)
• );

```

Numero	Rue	Code_Postal	Ville

```
CREATE TABLE adresses OF t_adresse;
```

```

CREATE TABLE ADRS OF T_ADRESSE(
CONSTRAINT PK primary key Numero,
CONSTRAINT CNN CHECK (rue IS NOT NULL)
);

```

Les types abstraits: exemple

- CREATE TYPE t_adresse AS OBJECT(
 - Numero NUMBER(4),
 - Rue VARCHAR2(20),
 - Code_postal NUMBER(5),
 - Ville VARCHAR2(20));

REF	Numero	Rue	Code_Postal
XXXXX			
YYYYYYY			
<u>ZZZZZZ</u>			

CREATE TABLE adresses OF t_adresse;

Les types abstraits: exemple

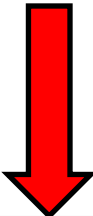
Insert into adresses values (**t_adresse**(15,'rue',1234,'ville'));

Insert into adresses values (15,'rue',1234,'ville');

Select * from adresses;

Select **REF(d)**
from adresses **d**;

Select **REF(d), d.***
from adresses **d**;



REF	Numero	Rue	Code_Postal
XXXXX			
YYYYYYY			
<u>ZZZZZZ</u>			

Les types abstraits: exemple

Insert into adresses values (**t_adresse**(15,'rue',1234,'ville'));

Insert into adresses values (15,'rue',1234,'ville');

Select **VALUE(d)** from adresses **d**;

t_adresse(15,'rue',1234,'ville');

Select **VALUE(d).rue** from
adresses **d**;

REF	Numero	Rue	Code_Postal
XXXXX			
YYYYYYY			
<u>ZZZZZZ</u>			

```
SQL>  
SQL> declare  
2   VADR T_adr;  
3   begin  
4   VADR:=NEW T_ADR('r',11,'v','p');  
5   insert into tab_adrs values(VADR);  
6   commit;  
7   end;  
8   /
```

Procédure PL/SQL terminée avec succès.

Les types abstraits: exemple

```
DECLARE
    ADR          T_ADRESSE;
    REFADR       REF T_ADRESSE;
BEGIN
    SELECT REF(D), VALUE(D) INTO REFADR, ADR
    FROM ADRESSES D
    WHERE D.RUE='sarue';
END;
/
```

Les types abstraits: exemple

```
• CREATE OR REPLACE TYPE t_personne AS OBEJCT(  
•     PL          NUMBER(4),  
•     Nom         VARCHAR2(20),  
•     Adresse     t_adresse,  
•     .....  
•     Salaire     NUMBER(10,2)  
• );
```

PL	NOM	Adresse				Salaire
		Numero	Rue	Code_Postal	Ville		

```
CREATE TABLE Employes OF t_personne;
```


Les types abstraits: exemple

- Insert into Employes values
(t_personne(11,'Zaatar',t_adresse(15,'rue',1234,'ville'),
...,1000000))
- Select * from Employes;
- Select Adresse from Employes;
- Select d.adresse.rue, d.adresse.ville
- From Employes d;

Les types abstraits: exemple

- CREATE OR REPLACE TYPE t_personne AS OBJECT(
 - PL NUMBER(4),
 - Nom VARCHAR2(20),
 - Adresse REF t_adresse,
 -
 - Salaire NUMBER(10,2));

```
CREATE TABLE PERSONNES OF t_personne(  
CONSTRAINT IREF ADRESSE SCOPE IS ADRESSES  
);
```

Les types abstraits: exemple

```
CREATE TABLE PERSONNES OF t_personne;
```

PL	NOM	Adresse	Salaire

Adresse			
Numero	Rue	Code_Postal	Ville

```
CREATE TABLE OF t_adresse;
```

Type abstrait: constructeur

- CREATE TABLE Ob_Employe OF t_personne;

INSERT INTO Employes VALUES (

t_personne(13,'toto',

(select REF(d) from adresses d where d.ville='saville'),

...

100000)

);

Les types abstraits: exemple

	Adresse			
REF	Numero	Rue	Code_Postal	Ville
324FER546482T				

PL	NOM	Adresse	Salaire
zaatar	toto	324FER546482T		

```
INSERT INTO Employes VALUES ( t_personne(13,'toto',
(select REF(d) from adresse d where d.ville='saville'),
...
1000000);
```

SELECT Deref(ADRESSE) FROM EMPLOYES;

SELECT D.ADRESSE.RUE FROM EMPLOYES D;

Les types abstraits (ADT)/ Conclusion

- Permet à l'utilisateur de définir lui-même ses propres types
- Pas de vraie identité d'objet mais des pointeurs invariables
- Le développeur gère lui-même les références des objets
- Navigation à travers les références au lieu de la jointure

LE MODELE RELATIONNEL ETENDU LES METHODES DE TYPE ABSTRAIT

SPECIFICATION DE TYPE

DECLARATION DES ATTRIBUTS

DECLARATION DES SIGNATURES
DES METHODES

CORPS DU TYPE

IMPLEMENTATION DU CORPS
DES METHODES

```
CREATE OR REPLACE TYPE BODY ttype AS
.....
.....
END;
/
```


Fonction ou procédure (PL/SQL, Java, C...)

Surcharge possible

Trois types

MEMBER (s'applique aux objets d'une table)

STATIC (s'applique au type seul) (DELETE et INSERT)

CONSTRUCTOR (création d'instances d'objets non persistants)

MAP (la position d'un objet dans l'ensemble des objets)

ORDER (comparaison de deux objets: SELF et un Objet passé en argument)

Appel

Objet de la table ou temporaire

PL/SQL (bloc, procédure, fonction, méthode)

Requête SELECT (pour les méthodes de type fonction)

Programme C, Java...

Implicite à la création d'un objet (constructeur)

Invoquer une méthode MEMBER

Principe pour récupérer un objet d'une table objet:
charger l'objet appelant de la table dans un objet temporaire par la fonction **VALUE**
puis invoquer la méthode sur
cet objet

Appel d'une procédure:

SELF → *obj.methode(paramètres);*

Appels d'une fonction:

SELECT a.methode(paramètres) FROM table a ...

resultat = obj.methode(paramètres);

EXEMPLE DE METHODES

```
--creation de type adresse
create or replace type t_adr as object(
    rue          varchar2(20),
    mais         number(2),
    ville        varchar2(20),
    pays         varchar2(20),

    -- le constructeur avec tous les arguments est crée par défaut
    -- si vous le créez, ca passera à la compilation
    -- mais provoquera une erreur à l'execution pour ambiguité

    constructor function t_adr(r IN varchar2, m IN Number, v IN varchar2) RETURN SELF AS RESULT,
    constructor function t_adr(r IN varchar2, m IN Number) RETURN SELF AS RESULT,
    MAP MEMBER FUNCTION COMPARE_ADR(ADR IN T_ADR) RETURN VARCHAR,
    member procedure modif_rue (REF_OBJ IN REF T_ADR,r IN varchar2),
    static procedure ajouter_adr(r IN varchar2, m IN number, v IN varchar2, p IN varchar2),
    static procedure ajouter_adr(A IN T_ADR),
    static procedure supprimer_adr(r IN varchar2, m IN number),
    -- on peut supprimer une objet par une méthode member
    member procedure sup_obj_direct
);
/
```

--codage des méthodes

create or replace type body t_adr as

constructor function t_adr(r IN varchar2, m IN Number, v IN varchar2) RETURN SELF AS RESULT IS
BEGIN

SELF.rue:=r;
SELF.mais:=m;
SELF.ville:=v;
SELF.PAYS:='maroc';
RETURN;

END t_adr;

constructor function t_adr(r IN varchar2, m IN Number) RETURN SELF AS RESULT IS
BEGIN

SELF.rue:=r;
SELF.mais:=m;
SELF.ville:='rabat';
SELF.PAYS:='maroc';
RETURN;

END t_adr;

```
MAP MEMBER FUNCTION COMPARE_ADR(ADR IN T_ADR) RETURN VARCHAR IS  
BEGIN
```

```
    RETURN T_CHAR(SELF.RUE || SELF.MAIS || SELF.VILLE || SELF.PAYS);  
END COMPARE_ADR;
```

```
member procedure modif_rue (REF_OBJ IN REF T_ADR, r IN varchar2) IS  
BEGIN
```

```
    UPDATE TAB_ADRS d  
    SET d.rue=r  
    WHERE REF(d)=REF_OBJ; --SELF.RUE=ADR.RUE and SELF.MAIS=ADR.MAIS;  
END modif_rue;
```

```
static procedure ajouter_adr(r IN varchar2, m IN number, v IN varchar2, p IN varchar2) IS  
BEGIN
```

```
    insert into tab_adrs values (r, m, v, p);  
END ajouter_adr;
```

```
static procedure ajouter_adr(A IN T_ADR) IS  
BEGIN
```

```
    insert into tab_adrs values (A);  
END ajouter_adr;
```

```
static procedure supprimer_adr(r IN varchar2, m IN number) IS
    REF_ADR          REF T_ADR; -- LA REF DE L'OBJET A SUPPRIMER
BEGIN

    DELETE TAB_ADRS WHERE RUE=r AND MAIS=m;

END supprimer_adr;

-- tester suppression d'objet sans passer par une methode statique
member procedure sup_obj_direct IS
BEGIN
    delete tab_adrs
    where rue=self.rue and mais=self.mais and ville=self.ville and pays=self.pays;
END SUP_OBJ_DIRECT;

END; --body
/
```

LE MODELE RELATIONNEL ETENDU
LES COLLECTIONS: LES TABLEAUX IMBRIQUES
VARRAY

- Ensemble d'éléments ordonnés de même type.
Chaque élément occupe une place unique
- Utilisation: implémenter une relation n-aire avec une cardinalité fixée:
- Exemple 1: une personne de sexe masculin peut avoir au maximum 4 conjointes
- Exemple 2: un module peut être composé au maximum de
- 5 cours.

Tab_Employes

nom	prenoms				Adr_privees				Adr_publicues		
	1	2	...	max	1	2	...	max	1	...	max
	string	string	...	string	Tadr	Tadr	...	Tadr	REF Tadr	...	REF Tadr
Toto										...	

Tab_Adresses

rue	ville
rue1	Ville1
rue2	Ville2

Exemple1:

Un tableau d'un type classique

```
create type liste_prenoms as varray(5) of varchar2(20);
```

Exemple2:

Un tableau d'un type objet

```
create type T_adresse as object (  
    rue varchar2(20),  
    ville varchar2(20));
```

```
create type liste_adresses_privees as varray(5) of T_adresse;
```

Exemple3:

Un tableau d'un type de référence vers des objets

```
create type T_adresse as object (  
    rue varchar2(20),  
    ville varchar2(20));
```

```
create type liste_adresses_publicques as varray(5) of REF T_adresse;
```

Exemple d'utilisation:

```
create table table_employees (  
    nom                varchar2(20),  
    prenom              liste_prenoms,  
    adr_privees         liste_adresses_privees,  
    adr_publiques       liste_adresses_publiques  
);
```

Exemple d'insertion d'un VARRAY:

```
insert into table_employees values (  
'nom1',  
liste_prenoms('prenom11', 'prenom12', 'prenom13'),  
liste_adresses_privees(T_adresse('r', 'v'), T_adresse('x', 'y')),  
liste_adresses_publicues((select ref(d) from ....), (select ...))  
);
```

Remarque: même principe que pour les objets et les références

Exemple de sélection dans un VARRAY: **aplatisseur TABLE**

```
SQL> --- sélection des rues de la personnes
```

```
SQL> ---
```

```
SQL>
```

```
SQL> select e.rue from table_employees p, table(p.adr_privees) e;
```

```
RUE
```

```
-----
```

```
rue11
```

```
rue12
```


Exemple de modification dans un VARRAY: **ACCES GLOBAL**

```
SQL> update table_employes
```

```
2  set prenom=liste_prenoms('XXXXX')
```

```
3  where nom='nom1';
```

1 ligne mise à jour.

```
SQL> select prenom from table_employes;
```

PRENOMS

LISTE_PRENOMS('XXXXX')

Exemple de modification d'un prénom dans une case donnée dans un VARRAY:

```
SQL> declare
2  new_prenom liste_prenoms;
3  begin
4  select prenom INTO new_prenom
5  from table_employes where nom='nom1';
6  new_prenom(1):='ZZZZZZ';
7  update table_employes
8  set prenom=new_prenom
9  where nom='nom1';
10 end;
11 /
```

Procédure PL/SQL terminée avec succès.

SQL>

SQL>

SQL> select prenom from table_employees;

PRENOMS

LISTE_PRENOMS('ZZZZZZ')

Exemple d'utilisation de EXTEND: allocation d'une nouvelle place si la limite n'est pas atteinte

```
SQL> declare
  2 new_prenom liste_prenoms;
  3 begin
  4 select prenom into new_prenom
  5 from table_employes where nom='nom1';
  6 new_prenom.EXTEND;
  7 update table_employes set prenom=new_prenom
  8 where nom='nom1';
  9 end;
10 /
```

Exemple d'utilisation de EXTEND:

SQL>

SQL> ---verification de l'extension

SQL>

SQL> select prenom from table_employees;

PRENOMS

LISTE_PRENOMS('ZZZZZZ', **NULL**)

Les méthodes associées au type VARRAY

Fonction	description
EXISTS(X)	Retourne TRUE si le Xime élément de la collection existe
COUNT	Retourne le nombre d'éléments dans la collection
LIMIT	Retourne le nombre maximum d'éléments dans un VARRAY
FIRST	Retourne le premier indice de l'élément de la collection
LAST	Retourne le dernier élément de la collection
PRIOR(X)	Retourne l'élément avant le Xime élément de la collection
NEXT(X)	Retourne l'élément après le Xime élément de la collection
TRIM(X)	Supprime X éléments à partir de la fin de la collection
EXTEND EXTEND(X) EXTEND(X,Y)	Ajoute une ou plusieurs copies à partir du Yime élément de la collection. X est le nombre d'élément à copier

Exemple d'utilisation d'une fonction de VARRAY

```
SQL> declare
```

```
2 new_prenom liste_prenoms;
```

```
3 begin
```

```
4 select prenom into new_prenom
```

```
5 from table_employees where nom='nom3';
```

```
6 if new_prenom.EXISTS(2) then
```

```
7 insert into table_employees (nom) values ('nom4');
```

```
8 else insert into table_employees (nom) values ('nom5');
```

```
9 end if;
```

```
10 end;
```

```
11 /
```

Procédure PL/SQL terminée avec succès.

LE MODELE RELATIONNEL ETENDU
LES COLLECTIONS: LES TABLES IMBRIQUEES
NESTED TABLE

- Les nested tables sont des tables imbriquées
- Un nested table peut être un champs, un attribut, une variable ou une table
- Les STORE Tables sont des segments physique où sont stockées les instances des tables imbriquées

UTILISATION:

**Implémenter les associations n-aires de cardinalité
Quelconque**

EXEMPLE:

le nombre d'employés dans un département peut être
quelconque

Numdep	nomdep	Projets		
11	NTIC	numprojet	nomprojet	budget
		123	Wifi	10000
		18	BDR	1.000.000,89
13	Physique	numprojet	nomprojet	budget

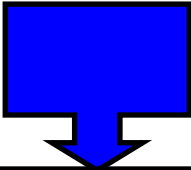
Table créée

```
CREATE TYPE t_projet AS OBJECT(  
    NumProjet    NUMBER(4),  
    NomProjet    VARCHAR2(20),  
    Budget       NUMBER(10,2)  
);  
/
```

```
CREATE TYPE t_tab_projets AS TABLE OF t_projet;  
/
```

```
CREATE TABLE Departement(  
    Numdep        NUMBER(6),  
    Nomdep        VARCHAR2(20),  
    Projets       t_tab_projets  
)  
NESTED TABLE projets STORE AS tab_des_projets;
```

Select p.* from Departement d, **table(d.projets) p;**

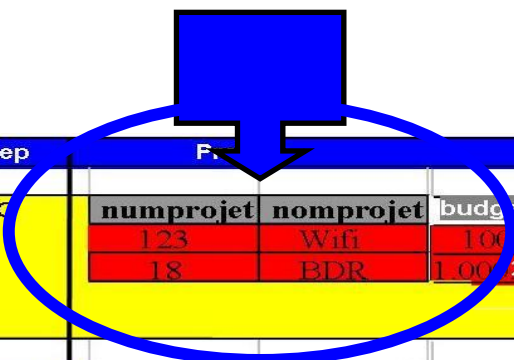


Numdep	nomdep	Projets		
11	NTIC	numprojet	nomprojet	budget
		123	Wifi	1000
		18	BDR	10002000,89
13	Physique	numprojet	nomprojet	budget

Projection sur la colonne projets en **fusionnant les tables imbriquées de tous les tuples**

THE (Select projets
from Departements
where numdep=11
);

Doit retourner une **SEULE** table
imbriquée au plus



Numdep	nomdep	F		
11	NTIC	numprojet	nomprojet	budget
		123	Wifi	10000
		18	BDR	1.002.000,89
13	Physique	numprojet	nomprojet	budget

Permet de récupérer la table imbriquée d'un tuple et
de manipuler la table ainsi récupérée comme une table classique pour:
Insert, update, select etc.

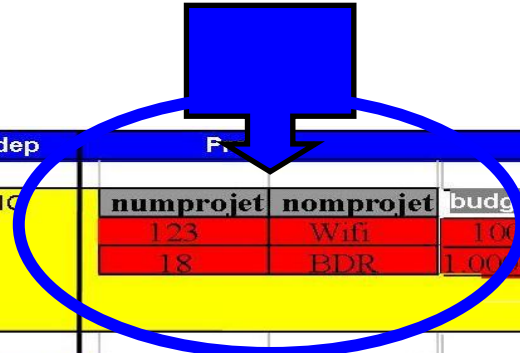
```
SELECT p.numprojet, p.nomprojet FROM  
      THE (SELECT d.projets FROM Departement d WHERE  
d.numdep=10) p WHERE p.budget>1000000;
```

```
INSERT INTO THE(SELECT d.projets FROM Departement d WHERE  
d.numdep=11) VALUES (  
t_projet(12,'MouvementPerpetuel',1000000000) );
```

```
UPDATE THE (SELECT d.projets FROM Departement d WHERE  
          d.numdep=13)  
SET budget=100  
WHERE numprojet=12;
```

A partir d'Oracle 10g

```
TABLE (Select projets  
from Departements  
where numdep=11  
);
```



Numdep	nomdep	F		
11	NTIC	numprojet	nomprojet	budget
		123	Wifi	10000
		18	BDR	1000000.89
13	Physique	numprojet	nomprojet	budget

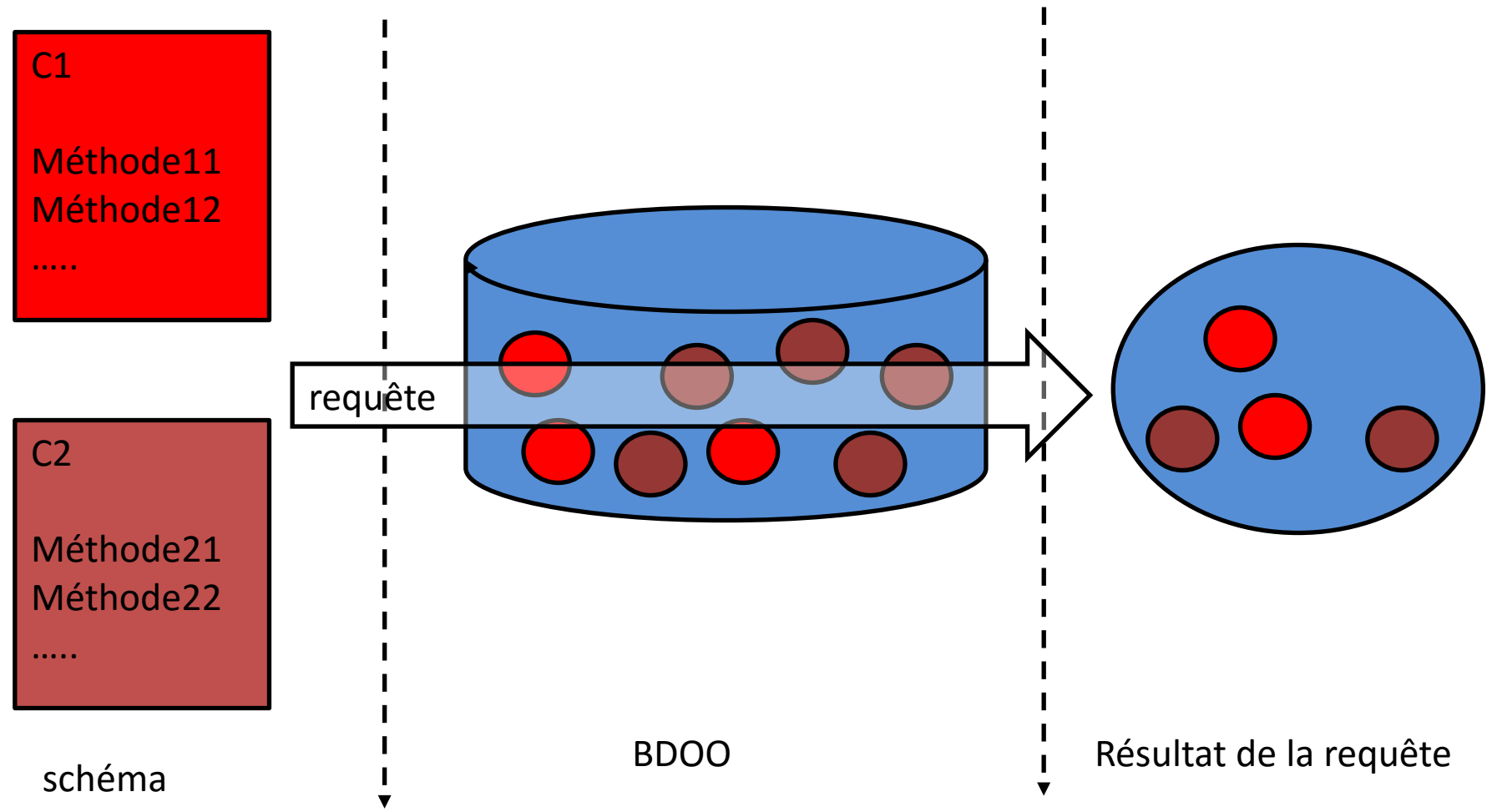
Doit retourner une **SEULE** table imbriquée au plus

Permet de récupérer la table imbriquée d'un tuple et de manipuler la table ainsi récupérée comme une table classique pour:
Insert, update, select etc.

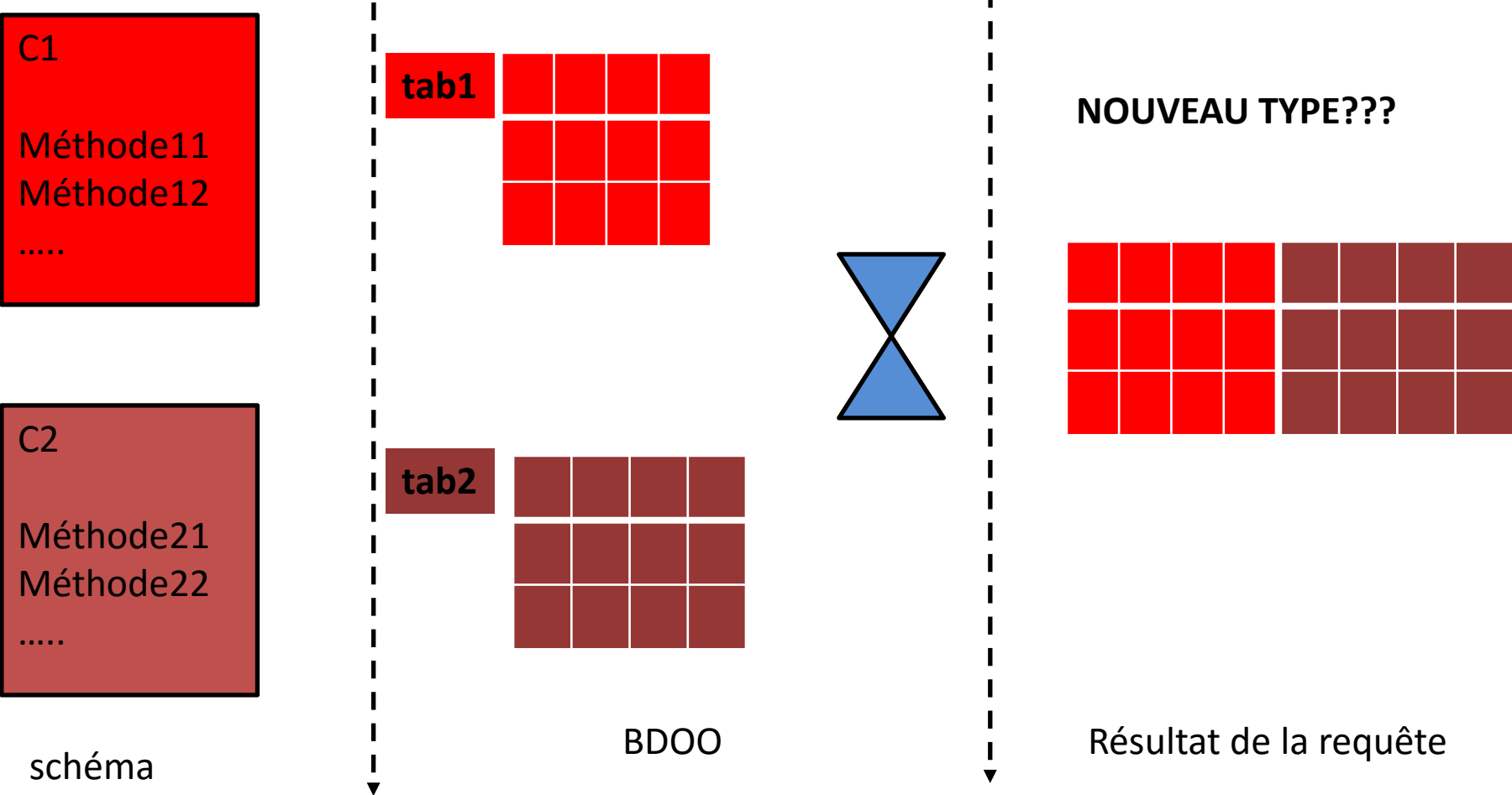
- Dans un bloc PL/SQL
- Select **LES_ELEVES** into **TAB_TEMP** from
- **TAB_TEMP.COUNT()**
- **TAB_TEMP.FIRST()**
- **TAB_TEMP.LAST()**
- **TAB_TEMP(ELE).<champs>**
- **TAB_TEMP1:=TAB_TEMP2;**

L'ENCAPSULATION

Encapsulation des données : SGBDOO



Encapsulation des données : SGBDR !!!!!



LES COLLECTIONS: TABLEAU

Tab_Employes

nom	prenoms				Adr_privees				Adr_publicues		
	1	2	...	max	1	2	...	max	1	...	max
	string	string	...	string	Tadr	Tadr	...	Tadr	REF Tadr	...	REF Tadr
Toto										...	

Tab_Adresses

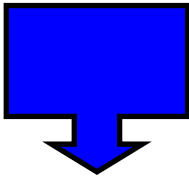
rue	ville
rue1	Ville1
rue2	Ville2

- Ensemble d'éléments ordonnés de même type. Chaque élément occupe une place unique
- **Utilisation: implémenter une relation n-aire avec une cardinalité fixée:**
- Exemple 1: une personne de sexe masculin peut avoir au maximum 4 conjointes
- Exemple 2: un module peut être composé au maximum de
- 5 cours.

LES COLLECTIONS: TABLE IMBRIQUEE

Numdep	nomdep	Projets		
11	NTIC	numprojet	nomprojet	budget
		123	Wifi	10001
		18	BDR	1.0002000,89
13	Physique	numprojet	nomprojet	budget

Select p.* from Departement d, **table(d.projets) p;**

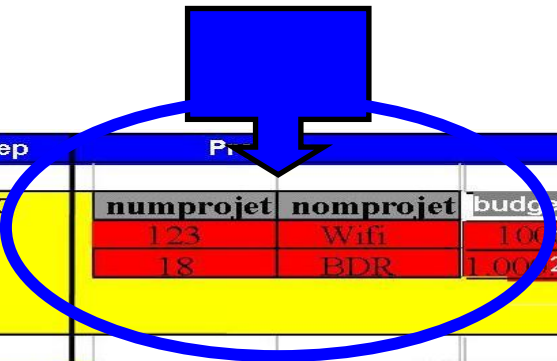


Numdep	nomdep	Projets		
11	NTIC	numprojet	nomprojet	budget
		123	Wifi	10000
		18	BDR	10002000.85
13	Physique	numprojet	nomprojet	budget

Projection sur la colonne projets en **fusionnant les tables imbriquées de tous les tuples**

```
THE (Select projets
      from Departements
      where numdep=11
    );
```

Doit retourner une **SEULE** table imbriquée au plus



Numdep	nomdep	Projets		
11	NTIC	numprojet	nomprojet	budget
		123	Wifi	10000
		18	BDR	1000000,89
13	Physique	numprojet	nomprojet	budget

Permet de récupérer la table imbriquée d'un tuple et de manipuler la table ainsi récupérée comme une table classique pour: **Insert, update, select etc.**

L'HERITAGE