SGBDOO et SGBDRO

Du relationnel à l'objet…

Plan du cours

- Modèle OO et SGBDOO
  - Pourquoi ?
  - Définition et concepts d'un SGBDOO
  - Requête OO
- Modèle RO et SGBRO
  - Introduction et comparaison OO
  - TAD
  - Les, collections, références et méthodes
  - Requête RO, notion d'Oid

Points faible du Relationnel

- Modèle
  - structure de données trop simple
  - pas de niveau conceptuel
  - intégration difficile avec les langages de programmation objets sans canevas logiciel
- Sur un SGBDR deux opérations
  - Join (lent)
  - Select (rapide)
- Cas des objets complexes
  - répartis sur plusieurs tables

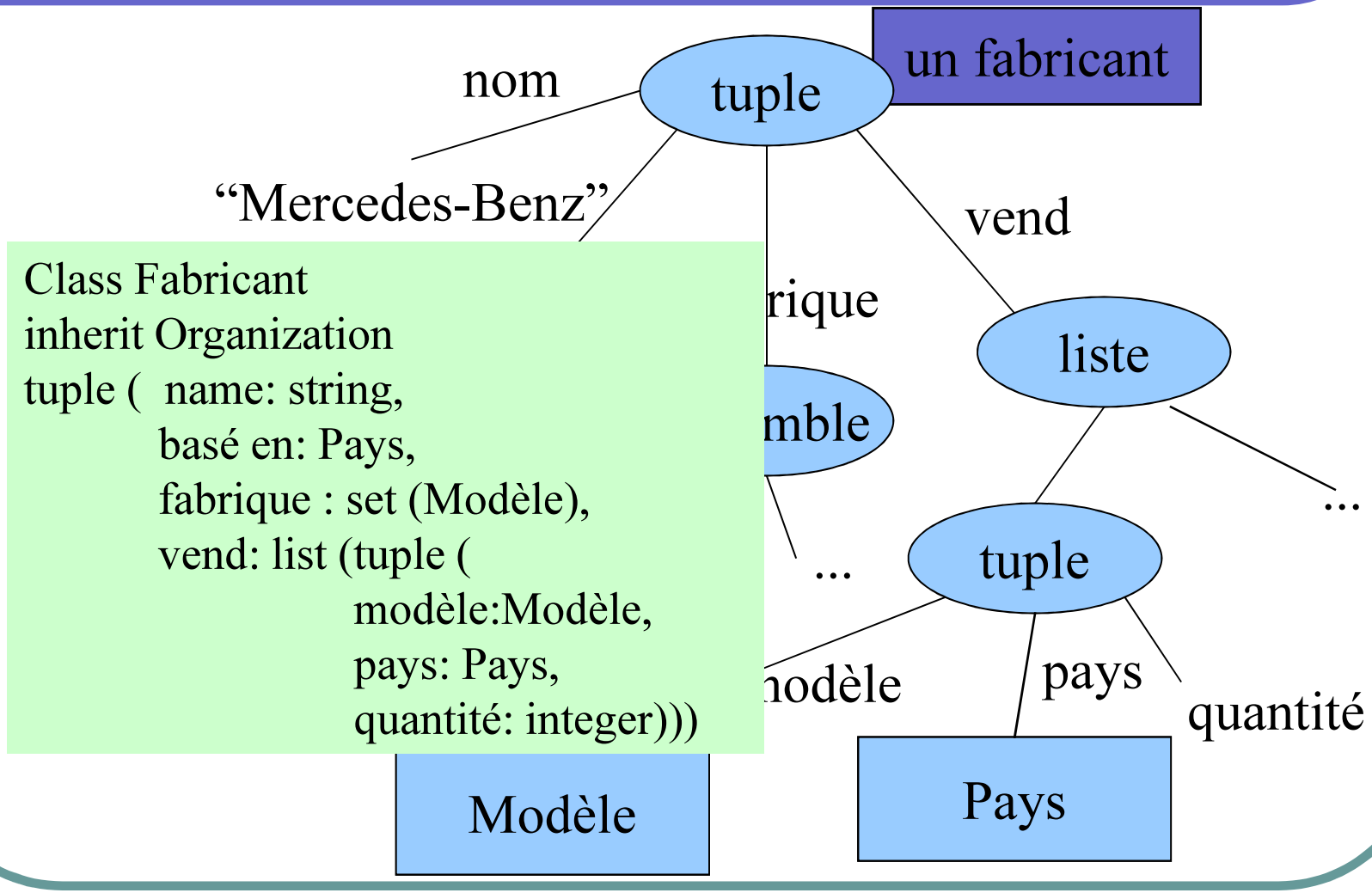
    $\Rightarrow$ beaucoup de jointures
  - mauvaises performances (sans optimisation)

Avantage de l'objet

- Classe = ensemble d'objets permanents de même structure ("population")
- Représentation fidèle du monde réel

  => liens sémantiques :
  - hiérarchie de généralisation = inclusion des populations
  - association
  - lien de composition conceptuel
- Requête = navigation dans un graphe d'objets

Un exemple d'objet complexe



SQL vs. OO

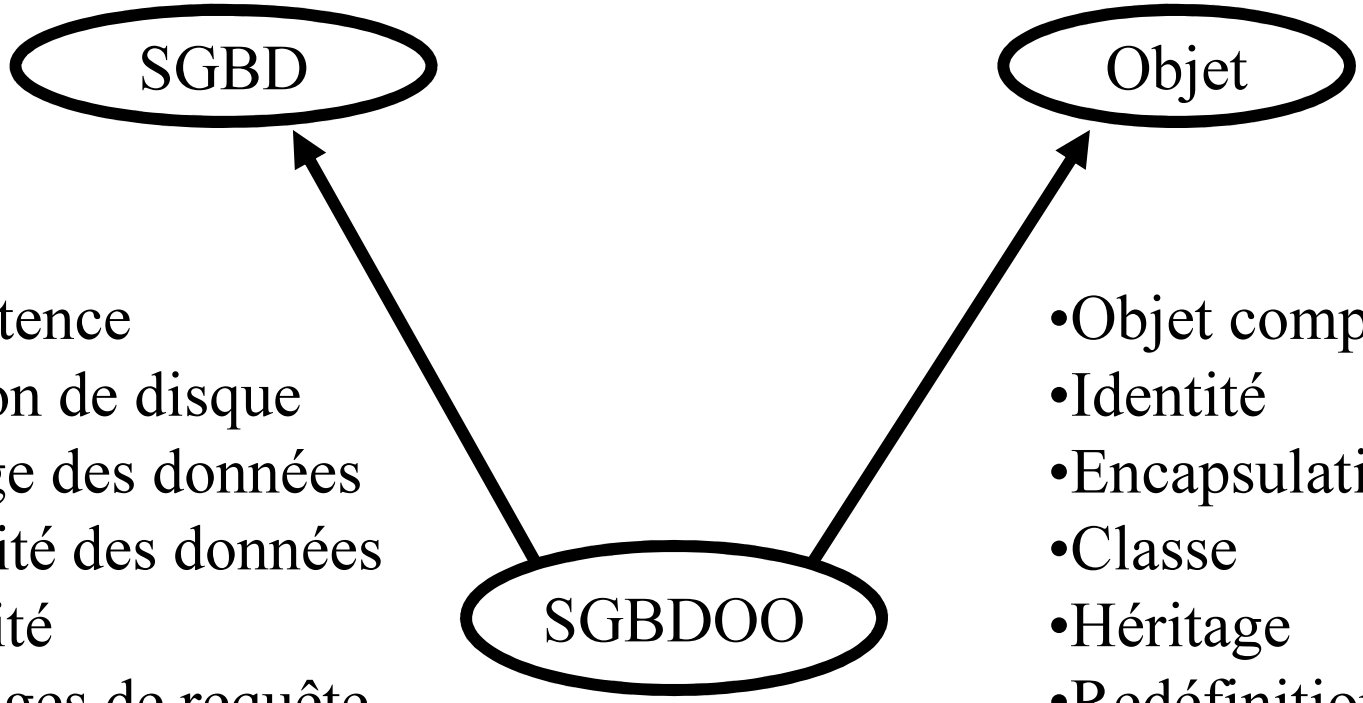
- "Modéles et quantité vendus par Mercedes en France"
  - en SQL

```
select Modèle.nom, ventes.quantité
from Fabricant, Modèle, ventes
where ventes.Fabric = Fabricant.ID.
    and ventes.Mod = Modèle.ID
    and Fabricant.nom = "Mercedes"
    and ventes.pays = « France"
```

  - en OQL

```
select c.vend.modèle.nom, c.vend.quantité
        from c in les_fabricants
        where c.nom = ”Mercedes”
                and c.vend.pays=“France”
```

SGBDOO : une définition



- Persistence
- Gestion de disque
- Partage des données
- Fiabilité des données
- Sécurité
- Langages de requête
- Indépendance logique/physique

- Objet complexe
- Identité
- Encapsulation
- Classe
- Héritage
- Redéfinition
- Complétude

Identité d'objets

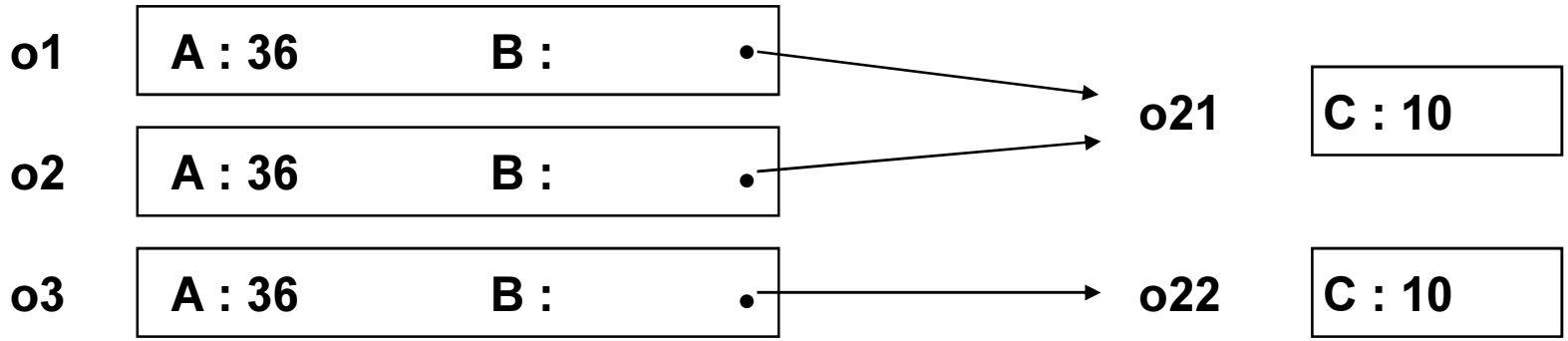
- Identifier les objets indépendamment de leur valeur
  - Chaque objet possède une identité qui ne peut être changée durant toute sa vie.
  - L'identification des objets est gérée par le système (allocation, …).
  - Les objets peuvent être partagés par d'autres objets.
  - Les objets peuvent être cycliques.
- oid (object identifier) géré par le SGBDOO
  - Unique, permanent, immuable
- objet = (oid, valeur)

=> test d'identité = =

test d'égalité = égalité en surface

= * égalité en profondeur

Tests d'identité / d'égalité



VRAI	FAUX	
o1.B = = o2.B	o1 = = o2	identité
o1 = o2	o1 = o3	égalité surface
o1 =* o3		égalité profonde
o21 =* o22	o21 = = o22	

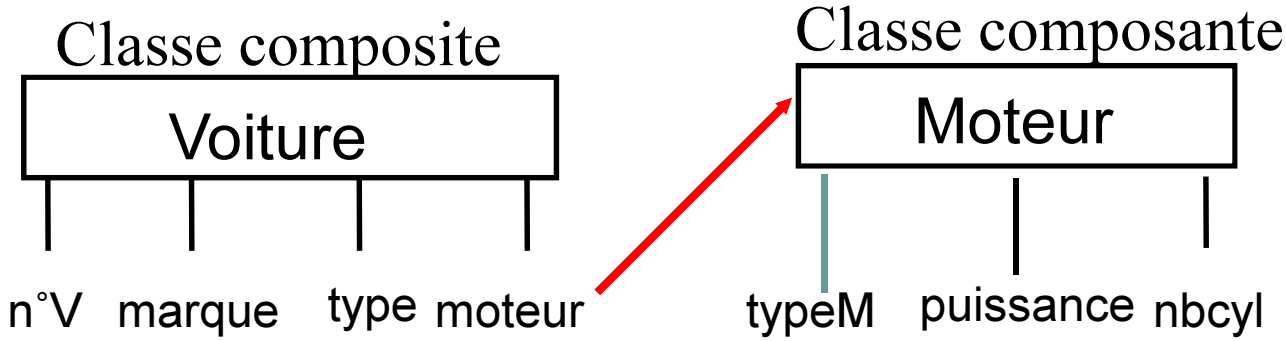
Notion de classe

- Une classe décrit les propriétés partagées par un ensemble d'objets similaires
  - propriétés statiques (structure de données)
  - propriétés dynamiques (méthodes)
- Une classe est composée de 2 parties :
  - l'interface : comment utiliser ses méthodes ?
  - l'implémentation : structure de données interne et code des méthodes
- Attention une classe n'a pas forcément de population (extension)

Exemple de classe

```
Class Employé
    type tuple ( nom : string,
                 emploi : Emploi,
                 expérience : integer,
                 photo : image )
        method
            public augmente_expérience (années: integer) : integer ,
            public affiche,
            public salaire : integer
    end;
```

Lien de composition



- **Class** Voiture

  tuple (n°V : **int**,
  marque : **char** 25,
  type : **char** 20,
  moteur : Moteur)

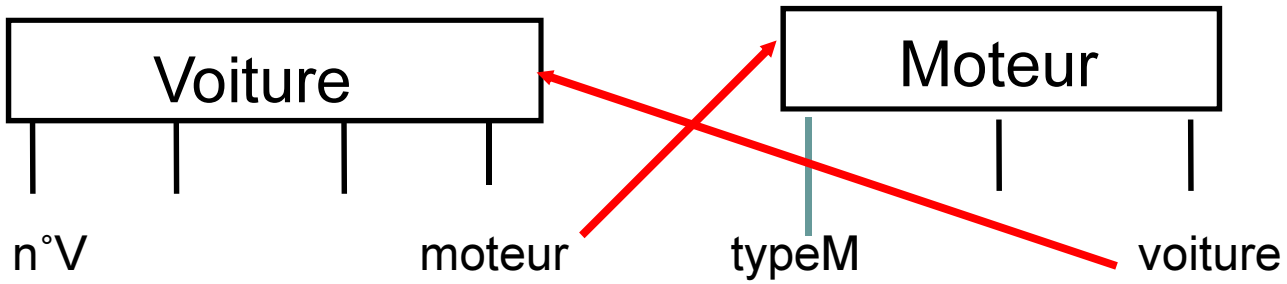
Class Moteur

tuple (typeM : **char** 20,
puissance : **int**,
nbcyl : **int**)

attribut référence
sa valeur = oid d'un moteur
ou NUL

Contraintes de composition

- objet composant partagé / non partagé
- objet composant dépendant / non dépendant
- lien inverse



- **cardinalités :**
  - **minimale, maximale**
  - **inverses**

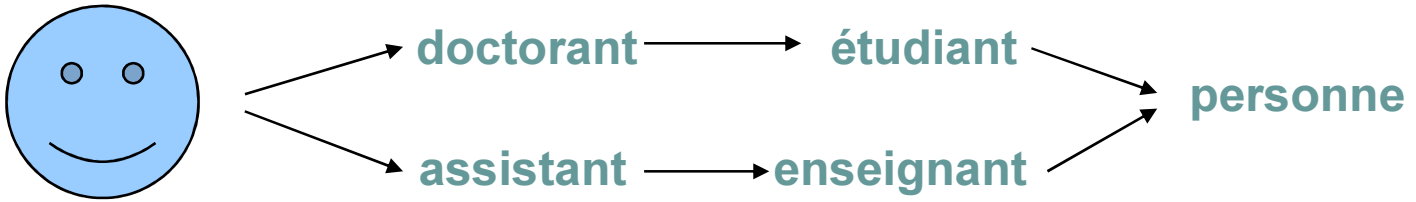
Hiérarchie de généralisation

- Objectifs :

  LPOO : héritage

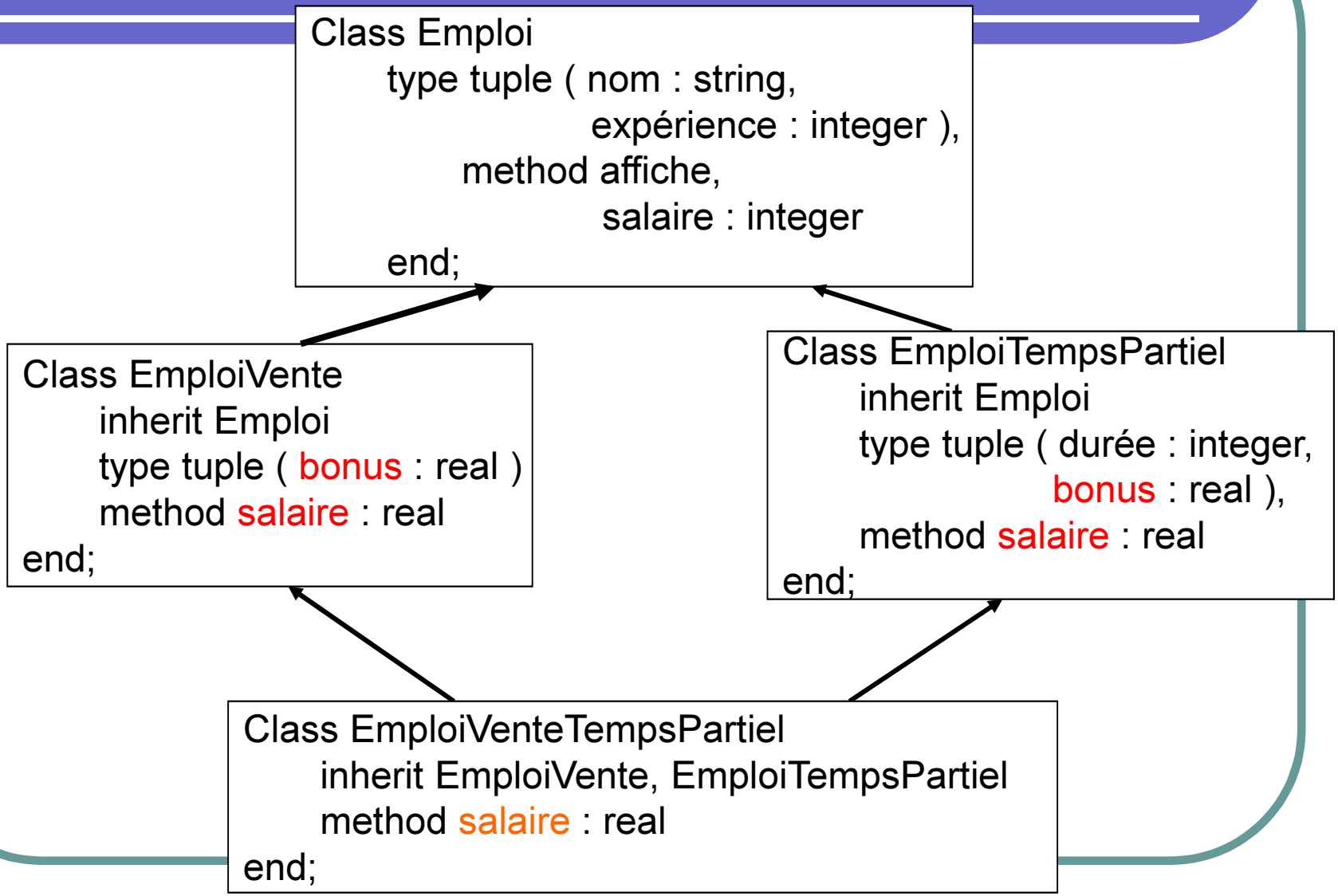
  BD : représentation du monde réel :

  différents points de vue sur le même objet



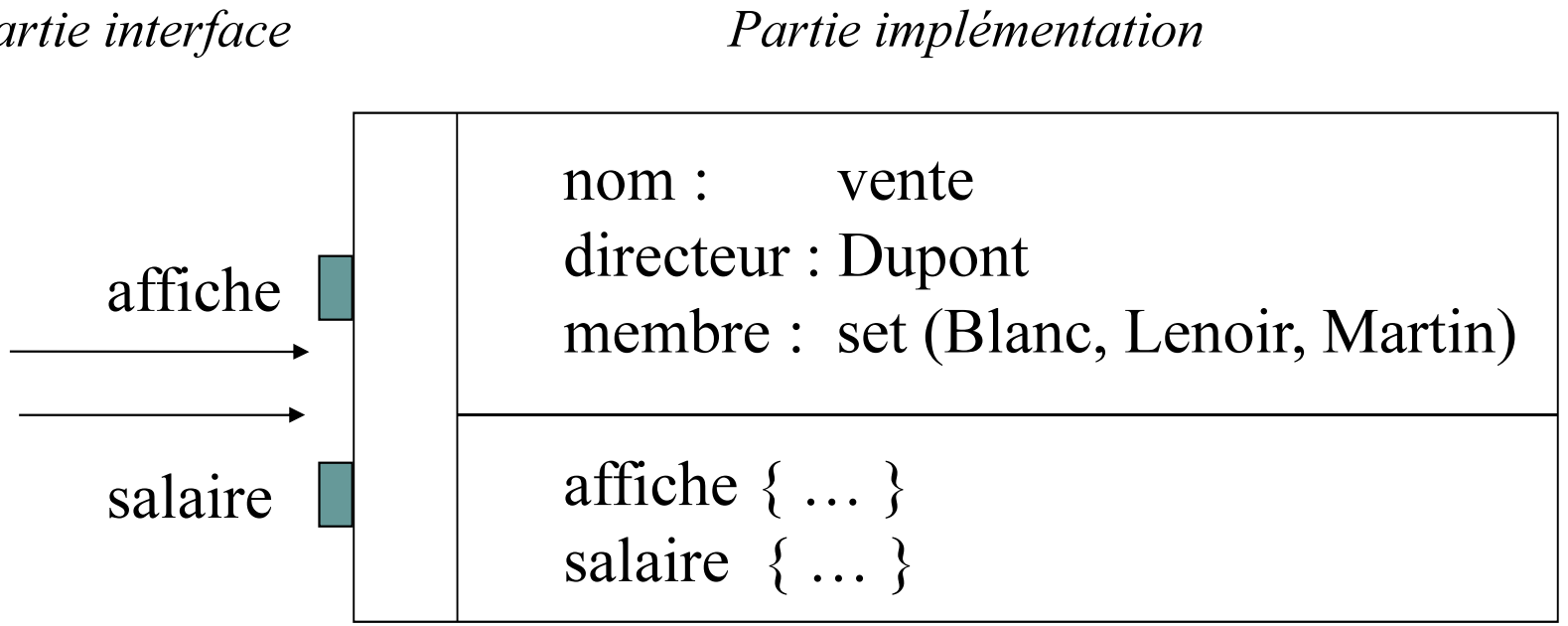
- **Lien de généralisation/spécialisation, IS-A:**
  - **inclusion de population**
  - **héritage des attributs et des méthodes**

Exemple de hiérarchie



Encapsulation

Les utilisateurs des objets (développeurs) ne doivent rien savoir à propos de leur implémentation : seule l'interface est publique



Redéfinition

- Avec une programmation classique :

```
for each emploi in LesEmplois do
    case of type (emploi)
        Emploi : salaire = salaire_standard (emploi)
        EmploiTempsPartiel : salaire = salaire_temps_partiel (emploi)
    end_case;
display (salaire);
```

- Avec une programmation objet :

```
for each emploi in LesEmplois do
    display (salaire);
```

Les collections en OO

- Collections d'objets :

  integer, n-uplets, nested collections ,….

  Exemples : LesNoms : unique set (string);

  class Département

  type tuple (nom : string;

  directeur : Employé,

  membres : set (Employé))

  LesDépartements : set (Département);

- Il est possible de définir des set, bag, list, array …
- Ces collections supportent toutes les opérations dédiées aux ensembles : union, intersection, parcours, sélection, jointure, …

Langage de requête

- Propriétés importantes :
  - supporter toutes sortes d'objets complexes
  - naviguer simplement à travers la BDO
  - tirer parti des méthodes des objets
  - construire des objets complexes comme résultat
  - permettre l'optimisation des requêtes
- Exemple de requête OQL

```
select tuple ( nom  : e.titre, salaire : e.salaire, directeur : dept.directeur.nom)
from    dept in LesDépartements, e in dept.membre
where dept.directeur.emploi.salaire > 100 000
   and e.emploi.nom = « Assistant »
```

RO: Modèle de données NF2 (*Non First Normal Form*)

- Structure des objets complexes dans un contexte relationnel.
- Cohabitation des notions de table (données en 1ère FN) et de données de type complexe (ou objet).
- Une table en NF2 peut contenir un attribut composé d'une liste de valeurs et/ou un attribut composé de plusieurs attributs.
  - Meilleur niveau conceptuel,
  - Plus besoin d'aplatir la réalité,
  - Moins de jointures.

Exemple : table Personnes

nom	{prenoms}	Date_naissance	{voitures}			…
	prenom		modele	annee	no	
Duran	Pierre	16-05-1963	2ch	1975	128	
	Paul		Mégane	2008	371	
	Jacques					
Piona	Marie	29-02-1994	Twingo	1999	17	
	Jeanne					

Concepts SGBDOO vs SGBDRO

Concepts SGBD

- Persistance
- Gestion du disque
- Partage des données
- Fiabilité
- Sécurité
- Langages de requête
- Indépendance Logique / Physique

- Objet (+ou-)
- encapsulation (+ou-)
- objet complexe
- identité d'objet
- Classe (+ou-)
- héritage
- redéfinition

- Persistance des objets complexes
- Gestion de large collection de données (+ou-)
- Langage de requête pour les objets

Le SGBDRO de Oracle

- Pas de classe.
  - Il faut créer un type abstrait de données (TAD), puis une table représentant ou utilisant ce type.
  - Implémentation des méthodes liée au TAD
- Héritage depuis oracle 9i
- Surcharge des noms de méthodes possible
  - 2 méthodes d'1 TAD avec le même nom et des paramètres différents => Oracle choisit à l'exécution.
  - Surcharge des opérateurs Oracle interdite.
- Encapsulation simulée à travers un package PL/SQL.

Concrètement

2 extensions par rapport au modèle relationnel :

- I - Les *TAD*
  (types abstraits de données),
  1. Composition de types,
  2. Collections,
  3. Références,
  4. Méthodes.

  } Types complexes

- II - Les *oid* via les pointeurs.

I – Les TAD

- Dans tout TAD peuvent être ajoutées des **méthodes** (procédures ou fonctions PL/SQL) pour manipuler les objets du TAD, mais **pas de contraintes**.
- Impossible de créer une instance d'un TAD directement, il faut créer et passer par une table !

  => Initialisation des instances impossible (pas de valeur par défaut associée à un TAD).
- Navigation à travers les instances des TAD avec la notation de chemin et un alias sur chaque table.

Instances de TAD

- Une table d'un TAD est une **table d'objets**.
  - Uniquement avec la commande :
    create table *nom_table* of *nom_TAD*;
  - Toute instance d'une telle table possède un oid unique, ce sont des n-uplets objet.
  - La portée de cet oid est globale.

- Attention,
  - Les autres tables (non directement construite sur un TAD) ne sont pas des tables d'objets.
  - Les instances des autres tables n'ont pas d'oid.
    - portée de l'oid locale à la table.

I - Les TAD :
1 – Composition de types

- Un TAD en RO peut être perçu comme :
  - Un nouveau type d'attribut simple défini par l'utilisateur (concerne 1 seul attribut),

  Ou
  - Une structure de données partagées.

A- TAD perçu comme un nouveau type d'attribut

- Cela concerne un seul attribut,
- Créé à partir d'un type d'Oracle,
- Permet d'enrichir les types prédéfinis d'oracle,
  - En ajoutant des méthodes de manipulation, avec la commande CREATE TYPE,
  - En ajoutant des contraintes de domaines,
    exemple : CREATE TYPE typemot IS varchar2 [NOT NULL]
    create table code (mot typemot(8), code typemot(25));

B- TAD perçu comme une structure partagée

- Types utilisables par un ou plusieurs autres types (composition) et par une ou plusieurs tables.
- Permet de construire une structure complexe,
  - Soit directement,
  - Soit indirectement.

TAD perçu comme une structure partagée

- Construction directe :
  1. Création d'un type,
  2. Création d'une table de ce type (=> table d'objets).
- Construction indirecte :
  1. Création d'un type *t* ayant au moins 1 attribut de type composé défini par l'utilisateur
     (i.e. : *t* utilise un TAD dans sa définition),
  2. Création d'une table de type *t* (=> table d'objets).

  Ou

  2. Création d'une table ayant au moins 1 attribut de type composé défini par l'utilisateur
     (=> **pas** table d'objets).

Composition directe de types

Create type tvoiture as object
(modele varchar2(15),
annee date, no integer)

/ *-- Remarque : après la création d'un TAD et uniquement dans ce cas, '/' remplace le ';' pour terminer un ordre PL/SQL.*

Create table voitures ***of*** **tvoiture**;

-- voitures est une table d'objets.

TAD Tvoiture :

Tvoiture		
Modele	Annee	No

Exemple de composition directe de TAD

TAD Tvoiture :

Tvoiture		
Modele	Annee	No

L'insertion des données peut se faire comme d'habitude en SQL :

Insert into voitures values ('2ch', '1975', 128);

Table voitures :

(ordre SQL :
select *
from voitures)

Modele	Annee	No
2ch	1975	128
Mégane	2008	371

Composition indirecte de types

```
Create type Tpersonne as object
        (nom varchar2(15), prenom varchar2(15),
        date_naissance date, voiture Tvoiture)
/
Create table personnes of Tpersonne;
    -- personnes est une table d'objets.
```

TAD Tpersonne :

Tpersonne					
Nom	prenom	Date_naissance	Voiture		
			modele	annee	No

Exemple de composition indirecte de TAD

A chaque TAD créé est associé un constructeur (même fonctionnement qu'en BDOO) du même nom que le TAD *(obligation de valuer tous les attributs).*

⇒ Insertion des données dans une table avec TAD :

Insert into personnes values (Tpersonne('Duran', 'Pierre', '16-05-1963', Tvoiture('2ch', '1975', 12)));

facultatif

obligatoire

Table personnes :

(ordre SQL :
select *
from personnes)

Nom	prenom	Date_naissance	Voiture		
			modele	annee	No
Duran	Pierre	16-05-1963	2ch	1975	12
Piona	Marie	29-02-1994	Twingo	1999	17

Construction indirecte de table

Création d'une table utilisant un TAD :

```
Create table personnes
        (nom varchar2(15), prenom varchar2(15),
        date_naissance date, voiture Tvoiture)
/ -- personnes n'est pas une table d'objets.
```

Insertion de données dans une table utilisant un TAD :

```
Insert into personnes values ('Duran', 'Pierre',
                '16-05-1963', Tvoiture('2ch', '1975', 12)  );
```

Valeurs par défaut

- Il est possible de définir des valeurs par défaut pour un TAD dans sa définition :

Create table personnes
(nom varchar2(15), prenom varchar2(15),
date_naissance date,
voiture Tvoiture DEFAULT voiture(‘2ch’,
‘1975’, 12))

TAD interdépendants

- Si références mutuelles des types, impossible de créer un TAD complet et correct sans que l'un des deux ne soit déjà créé => problème.
- Exemple :
  TAD *Tpersonne* utilise *Tvoiture*, et
  TAD *Tvoiture* utilise *Tpersonne*.
- Solution de Oracle : déf. de TAD incomplet.
  Syntaxe : create type nom_type
  / -- ni attribut, ni méthode déclarés.
- TAD complété plus tard, mais référençable dans l'état.

Les TAD :
2 – Les collections

- Exprimer qu'une personne possède plusieurs prénoms ou plusieurs voitures dans une même table.
- 2 possibilités de stocker des collections :
  1. Varray : dans une valeur
  2. Nested Table : dans une sous-table

Personnes

Nom	liste-prénoms	age	voitures
'Liera'	('Pierre','Paul')	32	Id1
'Liera'	('Zoé','Marie')	27	Id2

1 valeur

1 table

ID	Tvoiture		
Id1	'2ch'	'1975'	12
Id1	'206'	'2000'	3
Id2	'2ch'	'1975'	12
Id1	'607'	'2002'	1

Varray vs. Nested Table

- Relation d'ordre entre les éléments de la collection
- Stockage
- Type des éléments

- Requête
- Manipulation
- Optimisation
- Imbrication de collections

Varray	Nested Table
oui	non
1 valeur ou BLOB	1 table
Type objet ou scalaire*	
avec l'opérateur TABLE	
non	oui
non	index
non (mais ruse possible, avec Références)	

*types prédéfinis Oracle, y compris REF

Stocker des collections liées

- Construction d'un Varray :
  1. Création du type de la collection, avec :
     create type *nom* as Varray (nb_max) of type_elt,
  2. Création du type contenant la collection, Tpersonne par exemple (ou création directe de la table)
  3. Création de la table de ce type complexe (Tpersonne).
- Manipulation d'un Varray :

  Stocké et manipulé comme une valeur atomique

  ⇒ En 1ère forme normale

  ⇒ Manipulation de la liste, pas des éléments

  ⇒ Pas de jointure

Exemple de collections liées

Plusieurs prénoms pour une personne :

```
Create type liste-prenom as Varray(10) of varchar2(15)
-- Varray(10) : Varray est le mot-clé pour définir une collection et 10 le
     nombre maximum d'éléments de cette liste.
--  type des éléments de la collection indiqué après le mot-clé 'of'.
/
Create type or replace Tpersonne as object
          (nom varchar2(15), Date_n Date,
          prenoms liste_prenom, voiture Tvoiture)
/
Create table personnes of Tpersonne;
      -- personnes est une table d'objets.
```

Exemple de table avec collections liées

- La encore on utilise le constructeur pour les listes :

```
Insert into personnes values (Tpersonne('Duran',
    liste_prenom('Pierre', 'Paul', 'Jacques'),
        '16-05-1963', Tvoiture('2ch', '1975', 12)   );
```

Table personnes :
(ordre SQL :
select *
from personnes)

Nom	{prenoms}	Date_naissance	Voiture		
	Liste_prenom		modele	annee	No
Duran	Pierre	16-05-1963	2ch	1975	12
	Paul				
	Jacques				
Piona	Marie	29-02-1994	2ch	1975	12
	Jeanne				

Et moins simple

- Une personne peut avoir plusieurs voitures.

nom	{liste_prenoms}	Date_naissance	{voitures}			…
	prenom		modele	annee	no	
Duran	Pierre	16-05-1963	2ch	1975	128	
	Paul		Mégane	2008	371	
	Jacques					
Piona	Marie	29-02-1994	Twingo	1999	17	
	Jeanne					

Stocker des collections libres

Construction d'une Nested Table :

1. Création du type de la collection, avec :
   create type *nom* as Table of type_elt,
2. Création du type contenant la collection puis 3, ou création directe de la table, par exemple Personnes.
3. Création de la table du type 2 (s'il y a lieu).

⇒ Génération automatique d'un identifiant superficiel Nested_Table_Id pour chaque collection

⇒ Possibilité de créer un index dans la Nested Table

Attention : Une Nested Table n'est pas une table objet

⇒ Pas d'oid associé à un élément de collection

Créer des collections libres

```
Create type Tvoiture as object
                    (modele varchar2(15),
                    annee date, no integer)
/
Create type ens_voiture as Table of Tvoiture
/
Create table Personnes (nom varchar2(15),
          prenoms liste_prenom,
          date_naissance Date,
          voitures ens_voiture);
   Nested table voitures Store As ToutesVoitures
```

Personnes

Nom	liste-prénoms	age	voitures
'Liera'	('Pierre','Paul')	32	Id1
'Liera'	('Zoé','Marie')	27	Id2

Nested_table_Id	Tvoiture		
Id1	'2ch'	'1975'	12
Id1	'206'	'2000'	3
Id2	'2ch'	'1975'	12
Id1	'607'	'2002'	1

index

Manipuler des collections libres

Une collection dans une Nested Table peut être

- Insérée,
- Supprimée,
- Mise à jour : les éléments dans la nested table peuvent être insérés, supprimés ou maj

```
insert into Personnes values('mila',
        liste_prenom('Karim', 'Amin'), '16-03-75',
        ens_voiture(Tvoiture('2ch', 1975, 128),
                Tvoiture('Mégane', 2008, 179)));
insert into Table(select p.voitures
                        from Personnes p where p.nom= 'mila')
        values ('206', '2000', 3)
```

Manipuler ses éléments

- Supprimer une collection libre :

  Update Personnes SET voitures = NULL where nom= ‘mila’
- Re-créer : avec le constructeur (sinon, aucune opération possible)
  - Créer un ensemble vide :
    Update Personnes SET voitures = ens_voiture() where nom= ‘mila’
  - Créer un singleton :
    Update Personnes
    SET voitures = ens_voiture(Tvoiture(‘Mégane’, 2008, 179))
    where nom= ‘mila’
- **Attention :** une opération sur 1 élément d’1 collection dans une Nested Table pose un verrou sur son propriétaire

  => 1 opération à la fois dans une collection !!!!!!

Requête avec collections

- Create type liste_voiture as Varray(10) of Tvoiture
  /
  Create table personnes (nom varchar2(15),
  prenoms liste_prenom,
  date_naissance Date,
  voitures liste_voiture);

  insert into personnes values('mila',
  liste_prenom('Karim', 'Amin'), '16-03-75',
  liste_voiture(Tvoiture('2ch', 1975, 128),
  Tvoiture('Mégane', 2008, 179)));

- On veut exécuter des requêtes sur les collections de voitures, telle No des voitures des personnes.

Parcourir une collection

- Pour parcourir des collections libres ou liées : voire **une** collection comme **une** table.

⇒ Utilisation de l'opérateur **TABLE**
(liste ou 'sous-table' -> table).

⇒ Puis utilisation d'un alias pour parcourir les tables ainsi obtenues.

⇒ Parcours de chaque collection de chaque n-uplet.
Ici, parcours (sélection) de chaque collection 'prénoms' et 'voitures' de chaque personne de la table personnes.

Exemple de parcours de listes

- Numéro des voitures des personnes :

```
Select v.No
   from personnes p, Table (p.voitures) v;
```

- Numéro des voitures de Piona :

```
Select v.No
   from personnes p, Table (p.voitures) v
   where p.nom = 'Piona';
```

Les TAD :
3 – Les références

- Référencer une voiture dans une table sans l'expliciter à chaque fois.
  - Attention, impossible de référencer une collection !
- Possibilité de référencer des données via des pointeurs.
  - Créer le TAD t dont on veut référencer les instances,
  - Créer la table contenant les instances de t,
  - Puis :
    - Créer le TAD qui référence t (mot-clé REF),
    - Créer la table associée.

  ou
    - Créer directement la table.

Exemple de références

Référencer la voiture d'une personne :

```
Create type Tvoiture as object (modele varchar2(15),
                                annee date, No integer)
/
Create table voitures of Tvoiture;
Create or replace type Tpersonne as object
        (nom varchar2(15),
        prenoms liste_prenom,
        -- Rq : impossible de référencer une collection !
        -- prenoms REF liste_prenom est illégal.
        date_naissance Date,
        voiture REF Tvoiture)
/
Create table personnes of Tpersonne;
```

Insertion avec références

- Pour insérer un n-uplet dans la table personnes, il faut référencer la voiture de cette personne dans la table voitures

⇒ Récupérer l'adresse de cette voiture pour la placer dans la table personne.

⇒ Utilisation de l'opérateur REF qui renvoie l'adresse d'un n-uplet dans une table (REF : n-uplet -> pointeur).

⇒ Insert into personnes values ('Duran',
liste_prenom('Pierre', 'Paul', 'Jacques'),
'16-05-1963',
(select **REF**(v) from voitures v
where v.modele = '2ch'));

Tpersonne			
Nom	{prenoms} Liste_prenom	Date_naissance	@voiture

Table voitures :

modele	annee	No
2ch	1975	128
Mégane	2008	371

Tvoiture		
modele	annee	No

Table Personnes :

Nom	{prenoms} Liste_prenom	Date_naissance	@voiture
Duran	Pierre	16-05-1963	
	Paul		
	Jacques		
Piona	Marie	29-02-1994	
	Jeanne		

Requêtes avec références

- Résultat de la requête : select * from personnes;

Nom	{prenoms}	Date_naissance	@voiture
	Liste_prenom		
Duran	Pierre	16-05-1963	03F43970135A39847C… -- adresse du pointeur. Ne veut rien dire !
	Paul		
	Jacques		

- Pour obtenir la valeur référencée :

⇒ Utilisation de l'opérateur DEREF (pointeur -> valeur)

Select **DEREF**(p.voiture), p.nom from personnes p
where *p.voiture.année* < 1990;

Exemple – suite : Création directe de la table

Référencer la voiture d'une personne :

```
Create type Tvoiture as object
        (modele varchar2(15),
         annee date, No integer)
/
Create table voitures of Tvoiture;
 -- voitures est une table d'objets.
Create table personnes
        (nom varchar2(15), prenoms liste_prenom,
        date_naissance Date, voiture REF Tvoiture);
 -- personnes n'est pas directement construite avec un
TAD, donc n'est pas une table d'objets.
```

Mise à jour avec références

- Soit la table suivante :
  Create type Tadresse as object (ville varchar2(15), rue varchar2(15))
  /
  Create table personnes
  (nom varchar2(15), prenoms liste_prenom,
  adresse Tadresse, date_naissance Date,
  voiture REF Tvoiture);
- Modifier l'adresse :
  update personnes p set p.adresse.rue = 'Comédie'
  where p.adresse.ville = 'Montpellier';
- Changer de voiture :
  update personnes p set p.voiture =
  (select REF(v) from voitures v where v.modele = 'Mégane')
  where p.voiture.annee = 1975;
- **Attention** : modifier une voiture à travers une personne est impossible !

Référencer du vide

- Pour tester l'existence d'une instance référencée :
  Utilisation du prédicat IS DANGLING.

- Exemple :
  update Personnes set voiture = NULL
  Where voiture IS DANGLING

Références croisées

- Comme d'habitude, on commence par créer un type vide pour être référencé dans un second type et on modifie le premier type ensuite. On crée les tables en dernier.

Create type Tvoiture /

Create type Tpersonne as object
(nom varchar2(15), prenoms liste_prenom, date_naissance Date, voiture **REF** Tvoiture) /

Create type Tvoiture as object
(modele varchar2(15), annee date, No integer,
personne **REF** Tpersonne) /

Create table voitures *of* Tvoiture;

Create table personnes *of* Tpersonne;

Imbrication de collections

- Impossible dans l'absolu, mais des tableaux fixes (Varray) peuvent être imbriqués grâce aux références.

```
Create type tpanne as object
                    (typep varchar2(20),
                    datep date, detail varchar2(200)); /
Create type tpannes as Varray(10) OF REF tpanne; /
Create type tvoiture as object
                    (modele varchar2(15), annee date, no integer,
                    sespannes tpannes); /
Create type tvoitures as Varray(10) OF REF tvoiture;/
REMPLACE "Create type liste_voiture as Varray(10) of Tvoiture"
Create table pannes OF tpannes;
Create table voitures OF tvoitures;
Create table personnes (nom varchar2(15),
          prenoms liste_prenom, date_naissance Date, voitures tvoitures);
```

Les TAD :
4 – Les méthodes

- Déclaration des méthodes (signature) insérée après les attributs dans la déclaration des TAD

Create type [or replace] nom_type as object
(att1 type1, …, atti typei,
MEMBER signature_methode1, …,
STATIC signature_methodej),

Avec

- MEMBER : méthode invoquée sur les instances du TAD,
- STATIC : méthode invoquée sur le TAD,
- signature_methodel : *nom_méthode* (var1 type_var1, …).
  - typel ou type_varl défini par l'utilisateur ou prédéfini
  - *nom_ méthode* différent de celui du TAD et de ses attributs.

Corps des méthodes

- Code des méthodes dans le corps du TAD.
- Syntaxe :
  Create or replace type body nom_type as …
- Contient uniquement le corps des méthodes.
- Si uniquement des attributs dans un TAD, corps inutile.
- Possibilité d'utiliser le paramètre SELF dans le corps des méthodes (idem qu'en OO)
  - Implicitement ou explicitement déclaré,
  - Si déclaration implicite :
    IN pour les fonctions, IN OUT pour les procédures.

Exemple de méthode

```
Create or replace type Tpersonne as object
        (nom varchar2(15),
        prenoms liste_prenom,
        date_naissance Date,
        voiture REF Tvoiture,
        MEMBER procedure ajoute_prenom (prenom_sup
                IN OUT Tprenom) )
   /
Create or replace type body Tpersonne as
   MEMBER procedure ajoute_prenom
                (prenom_sup IN OUT Tprenom)  IS
   Begin …..
   End;
End
   /
```

Appel de méthode

- Avec notion de chemin (comme pour les attributs)
- Chaînage des appels de méthodes possible
  - Exécution des méthodes de gauche à droite,
  - ⇒ Pas d'appel à droite d'un appel de procédure
  - ⇒ Si chaînage, la 1ère fonction doit renvoyer un objet pouvant être passé à la 2nde.
- Exemple :

```
Update personnes p set
        p.prenoms =
        p.ajoute_prenom('Clémentine')
        where p.nom = 'Piona';
```

II – Les oid

- Chaque instance d'une table d'objets possède un **identificateur d'objet** qui :
  - Identifie l'objet stocké dans ce n-uplet de façon unique,
  - Sert à référencer l'objet (seul oid de portée globale).
- Comme en OO, utilisation des chemins (notation pointée) pour naviguer à travers les objets référencés.
- Rappel :
  - Table d'objets : uniquement créée avec la commande ''create table nom_table of nom_tad;''.

Rappel : Extraction d'un oid

- Utilisation de la primitive REF.
- Exemple (avec 1 voiture par personne):

```
Update personnes p SET
    p.voiture = (select REF(v) from voitures v
                                where v.modele='2ch')
    where p.nom = 'Piona';
```

- Utilisation identique dans les requêtes, insertions et suppressions.
- Permet d'assurer l'intégrité référentielle.

Sélection d'un objet

- Avec select : récupération d'ensemble de valeurs des attributs d'une table.
- Si on veut récupérer un ensemble d'objet :
  - Utilisation de l'opérateur VALUE
  - Exemple :
    - SQL> Select * from voitures;
      '2ch' ¦ '1975' ¦ 128
      'Mégane' ¦ '2008' ¦ 371
    - SQL> Select value(v) from voitures v;
      Tvoiture('2ch', '1975', 128)
      Tvoiture('Mégane', '1998', 371)
  - Idem avec insert et update (valeur/objet).

LMD et TAD

- Comme d'habitude en SQL,
  - Insert, Update, Delete, Select.
- Avec en plus les fonctions :
  - VALUE,
  - REF,
  - DEREF
- Le constructeur associé à chaque TAD,
- Le prédicat IS DANGLING,
- La clause RETURNING pour PL/SQL,
  ajout à la fin d'un insert ou update de la clause :
  Returning ref(alias_table) into nom_var
- L'opérateur TABLE.

LDD et TAD

- Créer : deja vu ….
- Modifier : Alter type
  Seules modifications possibles : les méthodes (insertions, suppressions, modifications)
- Supprimer : Drop type
- Rien n'est changé pour la modification et suppression des tables (y compris les tables d'objets).

Notion de Hiérarchie: Héritage

- Création de hiérarchie de types d'objet
  - Clause UNDER pour définir le sur-type d'un sous-type
  - Clause NOT FINAL pour autoriser un type à devenir sur-type
- Exemple

```
CREATE TYPE T-Personne AS OBJECT
( ninseeNUMBER ,
nom VARCHAR(18) ,
prénom VARCHAR(18) ,
adresse VARCHAR(200) )
NOT FINAL /
```

```
CREATE TYPE T-Etudiant UNDER T-Personne
( etab VARCHAR(18) ,
cycle VARCHAR(18) )
```

Notion de Hiérarchie: instances

- Une table T-Personne peut recevoir des tuples de type T-Personne et T-Etudiant

```
CREATE TABLE LesPersonnes OF T-Personne ;

INSERT INTO LesPersonnes VALUES (T-Person
            (11111111, 'TTT', 'SSSS', 'Rue des oiseaux, 1, MMMM') );
INSERT INTO LesPersonnes VALUES ( T-Etudiant
            (22222222, 'Muller', 'Annie', 'Rue du marché, 22, RRRR', 'UJF', 'master'));
```

- Exemple de requête
  - **VALUE(objet) IS OF type**: test intéressant relatif aux relations sur-type/sous-type

```
SELECT p FROM LesPersonnes p WHERE VALUE(p) IS OF T-Etudiant
```

Renvoie les objets de LesPersonnes qui sont de type T-Etudiant