Chapitre 3: BASES DE DONNEES ORIENTEES OBJETS

Module: Bases de données avancées
Master STIC (Semestre 1)
Département Informatique, Université de GuGuelma 20**21**/20**22**

Agenda

- Introduction à l'ODMG:
  - contenu de la proposition;
  - architecture d'un SGBDO;
  - ...
- ODL:
  - exemple de type ODL.
- OQL:
  - objectifs;
  - accès à partir d'objets nommés;
  - sélection avec qualification;
  - ...
- Exercices

Bases de données orientées objets

Principes des SGBD OO

Nouvelles applications

- conception assistée par ordinateur
- production assistée par ordinateur
- génie logiciel
- systèmes d'informations géographiques
- systèmes multi-média
- recherche et intégration de données de la toile
- …

- Nouveaux besoins
  - objets structurés, volumineux
  - nouveaux types de données
  - transactions longues
  - …
  - développement des SI non satisfaisant

ODMG (Object Database Management Group)

Objectifs de l'ODMG:

Réaliser l'équivalent de la norme SQL pour les bases de données objets.

Permettre l'utilisation directe des types des langages objet.

Définir un modèle abstrait de définition de bases de données objet, mis en oeuvre par un langage appelé ODL (Object Definition Language).

Adapter le modèle à un langage objet particulier:
C++;
Smalltalk;
Java.

Proposer un langage d'interrogation: OQL (Object Query Language).

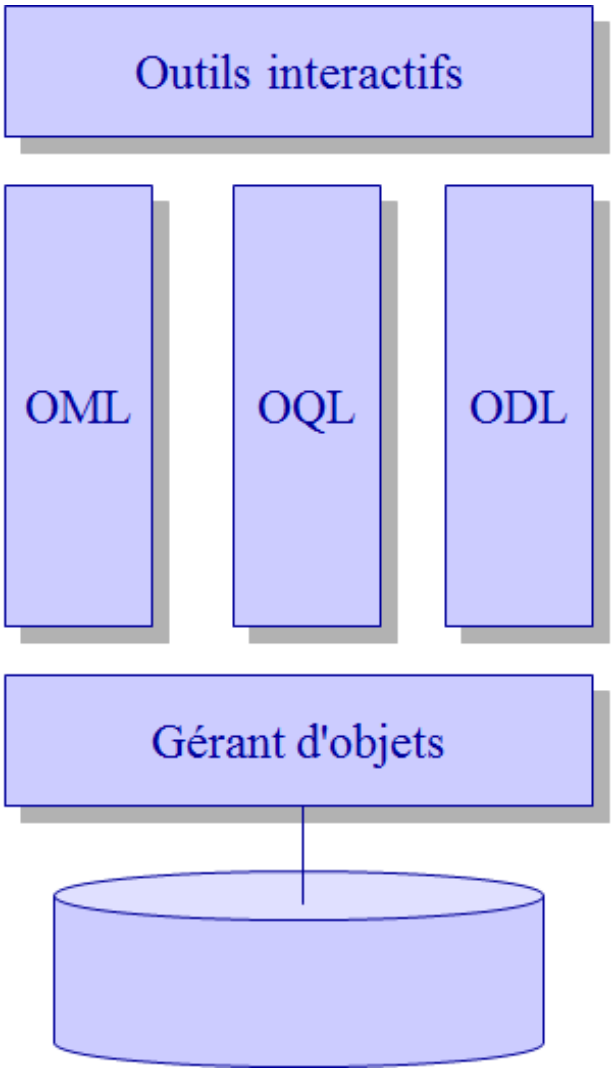
Contenu de la proposition

- **ODL - *Object Definition Language***
  Langage de définition de schéma des bases de données objet proposé par l'ODMG. (Equivalent des *DDL - Data Definition Language* des SGBD.)

- **OQL - *Object Query Language***:
  Langage d'interrogation de bases de données objet proposé par l'ODMG, basé sur des requêtes SELECT proches de celles de SQL.

- **OML - *Object Manipulation Language***:
  Langage de manipulation intégré à un langage de programmation objet permettant la navigation, l'intérrogation (OQL) et la mise à jour de collections d'objets persistants, dont l'OMG propose trois variantes: OML C++, OML Smalltalk, et OML Java.

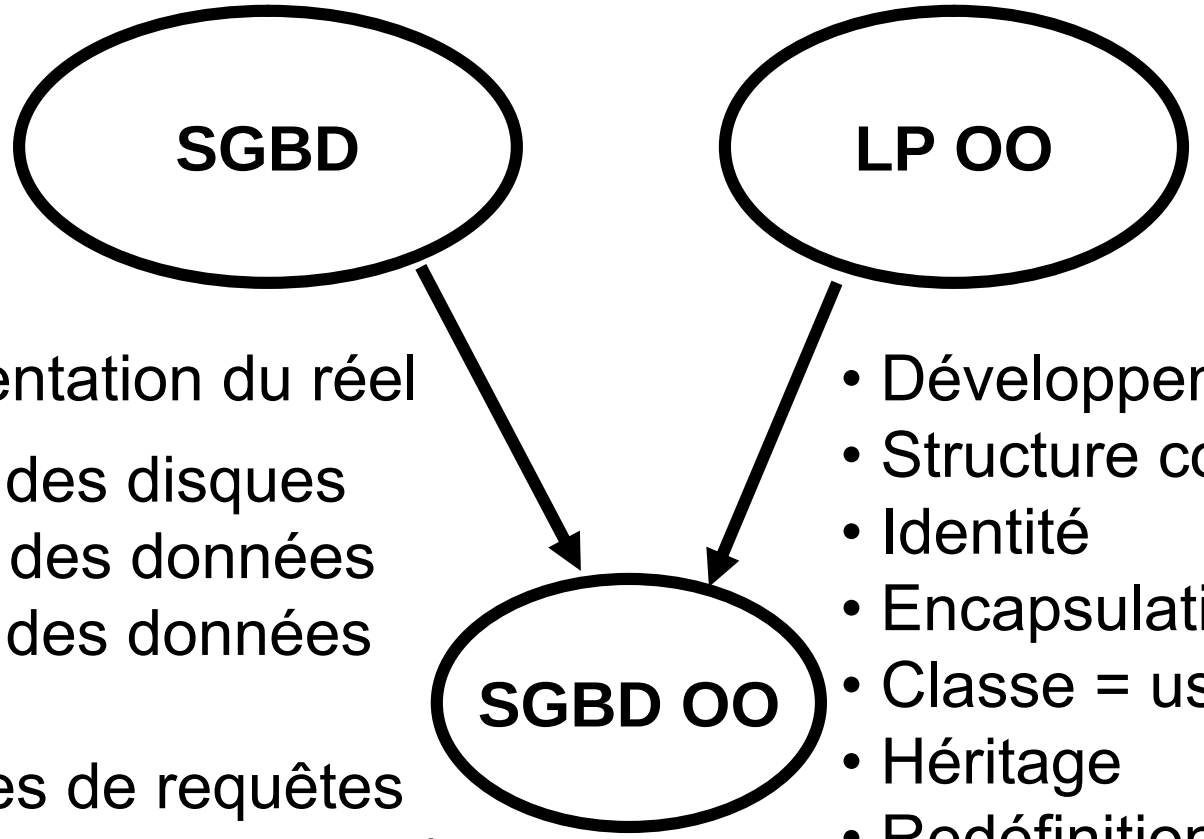
Architecture d'un SGBDO conforme à l'ODMG



- Editeur de classes
- Manipulateur d'objets
- Bibliothèques graphiques
- Débogueur, éditeur

- Persistance
- Identification
- Accès
- Concurrence
- Fiabilité
- Sécurité

SGBD OO = LPOO + BD



- Représentation du réel
- Gestion des disques
- Partage des données
- Fiabilité des données
- Sécurité
- Langages de requêtes
- Indépendance logique / physique

- Développement
- Structure complexe
- Identité
- Encapsulation
- Classe = usine
- Héritage
- Redéfinition
- Bibliothèques de classes

Intérêt d'un SGBD OO / SGBDR

C'est mieux qu'un SGBD relationnel :

- permet la manipulation d'objets à structure

  complexe
- interface compatible avec les LP-OO
- nouveaux types de données (image, son…)
- performances

Bases de données orientées objets

Modélisation

Diversité des modèles

- Norme ODMG
  mais de nombreux SGBDO ne la suivent pas.

- Ce cours définit :
  - les principes communs aux SGBD OO
  - les alternatives importantes

- Ce cours emploie une syntaxe tirée de celle d'ODMG

Concepts principaux

Monde réel	**BD OO**
objet	objet, classe d'objets
propriété	attribut méthode
lien	lien de composition binaire sans attribut orienté
représentation multiple	hiérarchie de généralisation/ spécialisation, héritage

OBJETS A STRUCTURE COMPLEXE

- Objectif : représentation directe des objets du monde réel
- Monde réel : Personne

  nom

  prénoms

  adresse (rue, n°, ville, codeNPA)

  enfants (prénoms, sexe, dateNais)

  …

- En relationnel : **4** relations, **N** tuples

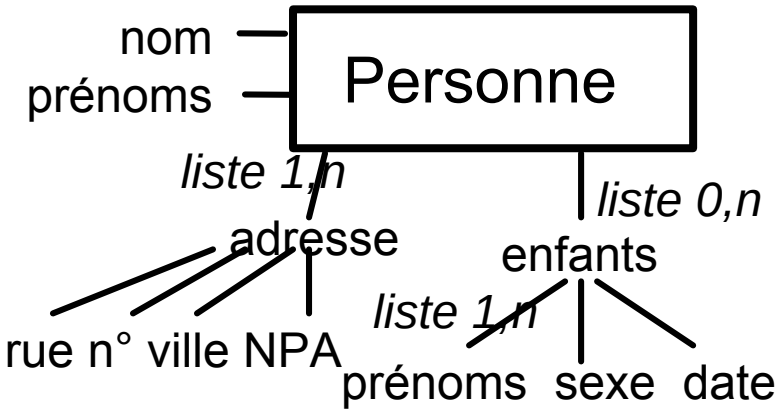
  Personne (n°, nom, adresse_rue, adresse_n°, adresse_ville, adresse_codeNPA)

  Personne_prénom (n°P, n°prénom, prénom)

  Personne_enfant (n°P, n°enfant, sexe, dateNais)

  Person_enfant_prénom (n°P, n°enfant, n°prénom, prénom)

Structure complexe



En OO : **un** seul objet

CLASS **Personne**

{ ATTRIBUTE **nom** : STRING ,

ATTRIBUTE **prénoms** : LIST STRING ,

ATTRIBUTE **adresse** : STRUCT **adr**

{ **rue** : STRING ,

n° : STRING ,

ville : STRING ,

codeNPA : INT }

ATTRIBUTE **enfants** : LIST STRUCT **enfant**

{ **prénoms** : LIST STRING ,

sexe : ENUM {'M', 'F'} ,

date : DATE }

}

Structure complexe (suite)

- Constructeurs de structure complexe :
  - attribut complexe : STRUCT
  - attribut multivalué => constructeur de collection
    - ensemble : SET
    - liste : LIST
    - multi-ensemble : BAG
    - tableau à une dimension : ARRAY

Types définis par l'application

- Les constructeurs de structure complexe servent à:
  - définir des **classes d'objets** à structure complexe
  - définir des **types de données** adaptés à l'application
    - type T-Adresse
    - types Point, Ligne, Polygone
    - types Image, Son …
- Comme les classes d'objets, les types de données définis par l'application ont :
  - une structure complexe
  - des opérations (méthodes)

Types de données - Exemple

```
TYPEDEF T-Adresse STRUCT
  { ATTRIBUTE rue : STRING  ,
    ATTRIBUTE n° : STRING ,
    ATTRIBUTE ville : STRING ,
    ATTRIBUTE codeNPA : INT }


CLASS Personne
  { ATTRIBUTE nom : STRING ,
   ATTRIBUTE prénom : LIST STRING ,
   ATTRIBUTE adresse : T-Adresse ,
   ATTRIBUTE enfants : LIST STRUCT enfant
        { prénoms : LIST STRING ,
         sexe : ENUM {'M', 'F'} ,
         date : DATE } }
```

OBJET AVEC IDENTITE

- Objectif : Identifier les objets indépendamment de leur valeur et de leur adresse (MC ou disque)

  =>? ? ? ? ? ? ? ? ? ? ? ? ? aux changements de valeur

  => insensibilité aux déplacements internes

- Chaque objet possède une identité propre qui ne peut être changée durant toute sa vie
- L'identification des objets est gérée par le système (allocation).
- Intérêt de l'identité d'objet
  - Représentation directe du monde réel
  - Permet de représenter des doubles
  - Moyen efficace pour référencer un objet

Identité en orienté objet

- oid (object identifier) géré par le SGBD OO
  - unique
  - permanent

- objet : (oid, valeur)

Lien de composition

CLASS Voiture

{ modèle : STRING ,
marque : STRING ,
type : STRING ,
moteur : Moteur }

CLASS Moteur

{ N° : STRING ,
puissance : FLOAT ,
nbCyl : INT }

Lien de composition / association

BD OO	Entité Association
■ Sémantique :	
"composition"	association générique
Voiture –> Moteur	Etudiant --inscription-- Cours
■ orienté	non orienté
■ binaire	n-aire

HIERARCHIE D'HERITAGE

- Objectif des LP OO : réutilisation (réduire le coût de développement)

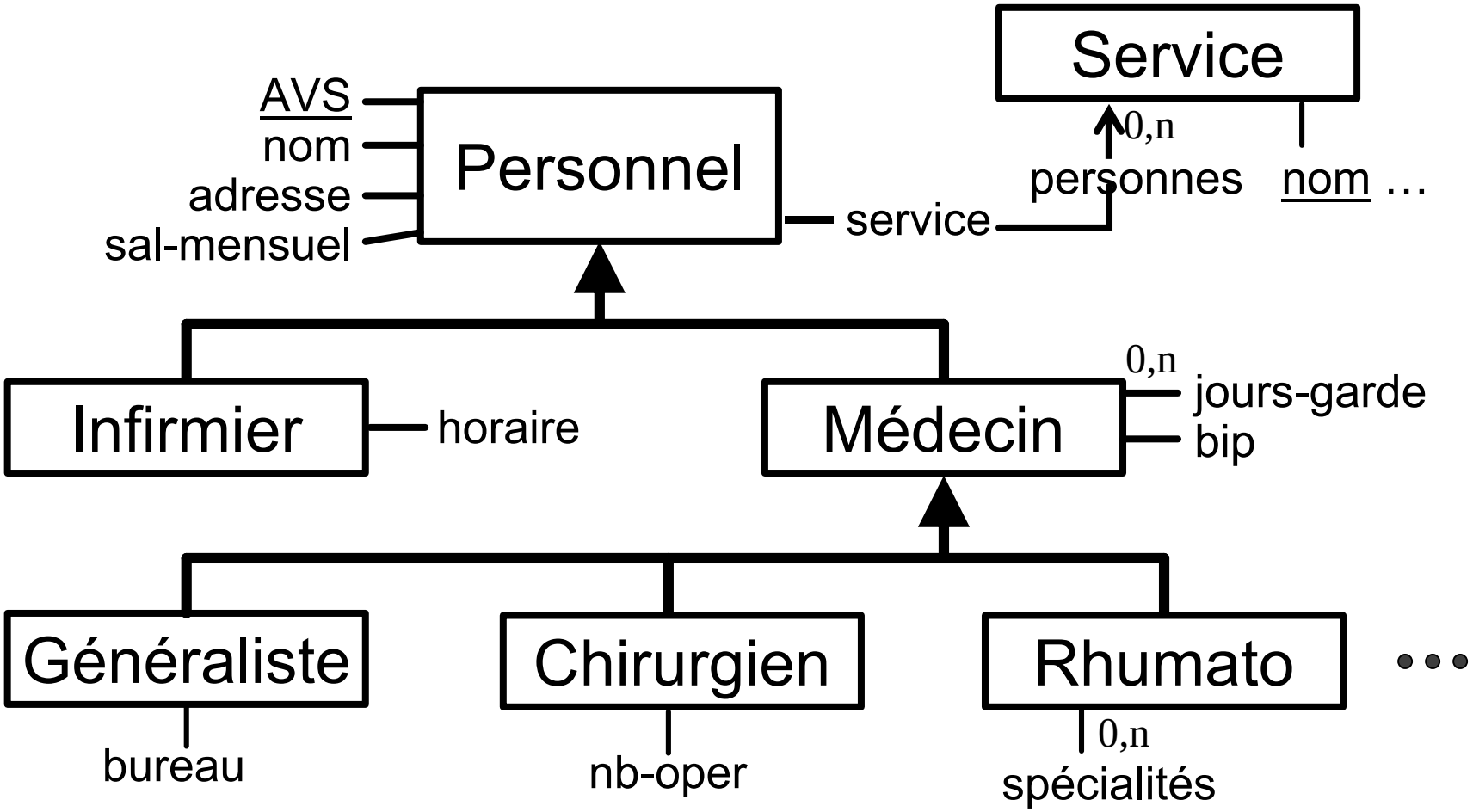
  ==> Héritage des propriétés

  Redéfinition des propriétés pour les adapter

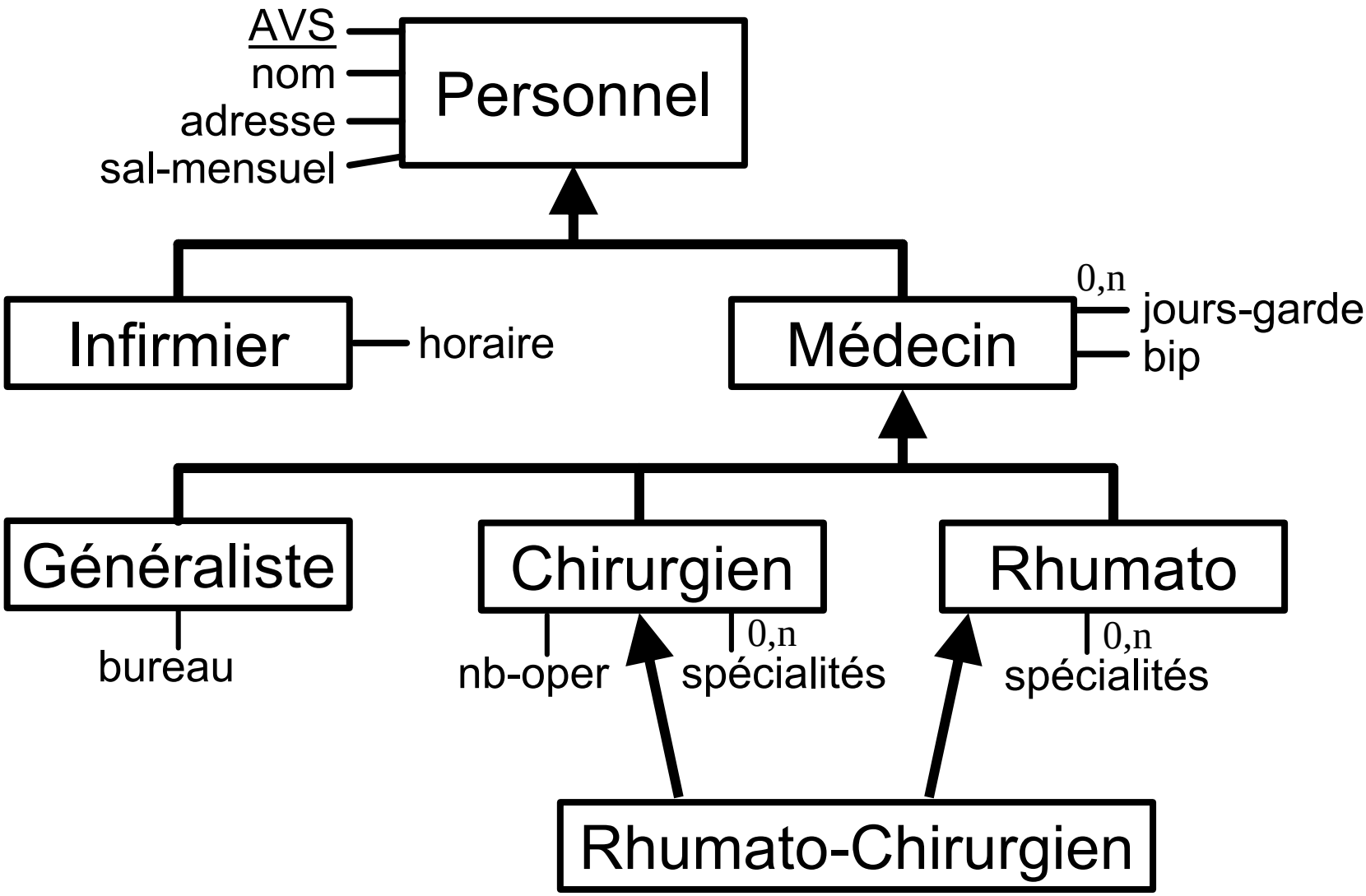
- Objectif des BD OO : représentions multiples du même objet

  - Mohamed est :
    - membre du personnel de l'hôpital
    - médecin
    - chirurgien
    - et en ce moment un patient

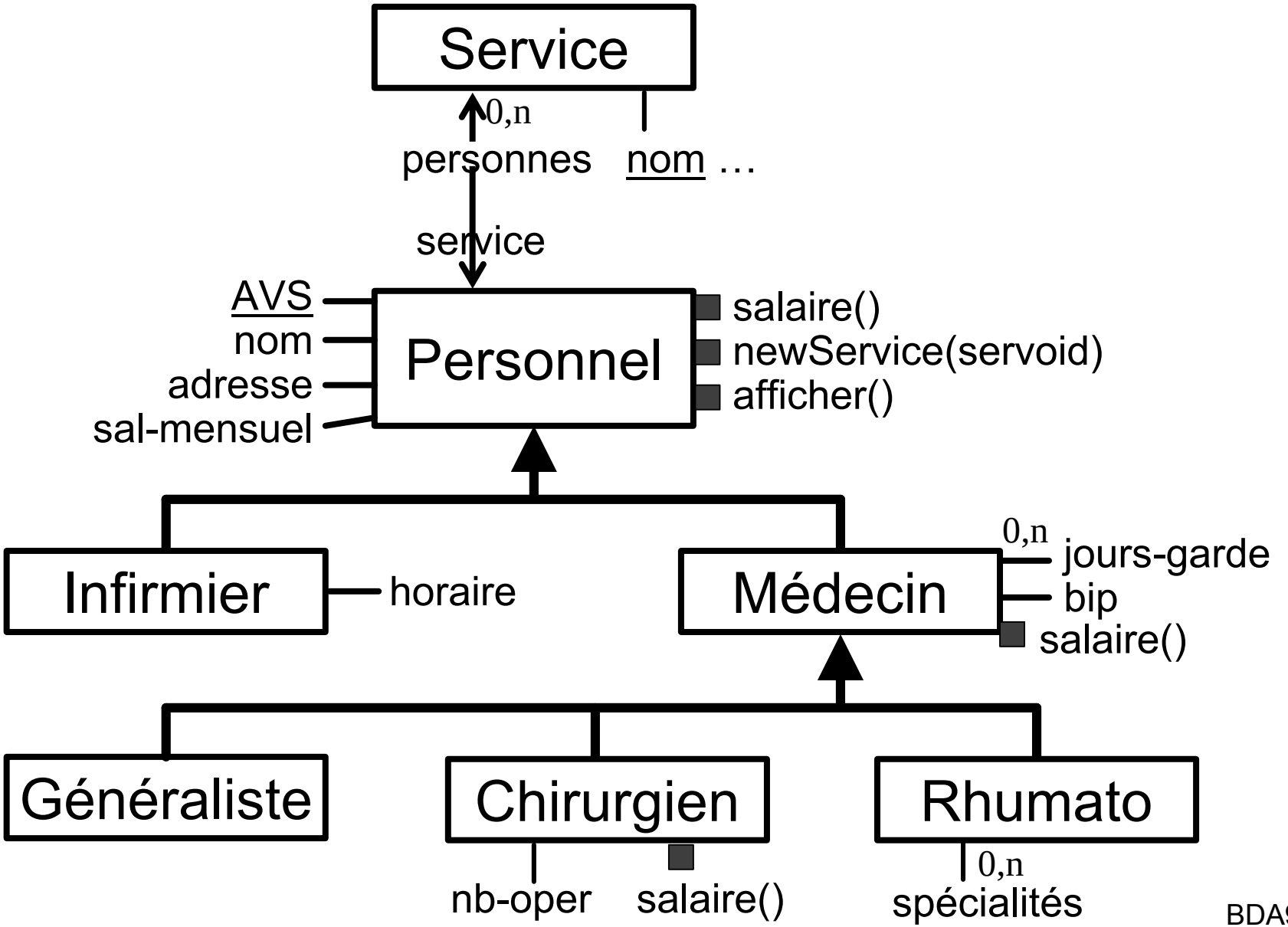
Exemple : le personnel d'un hôpital



Héritage multiple



Personnel d'un hôpital avec méthodes



Encapsulation

- Objectif : cacher l'implémentation des classes pour
  - faciliter la réutilisation des classes : il suffit d'en connaître l'interface
  - permettre l'évolution de l'implémentation des classes : si elle change, l'application doit seulement être re-compilée
- Principe : depuis l'extérieur de l'objet seules les signatures de ses méthodes sont visibles
- Implantation cachée
  - structure des objets
  - code des méthodes

Exemple d'encapsulation

CLASS Personnel

- Interface visible

  INT salaire()

  VOID newService(servoid : Service)

  VOID afficher()

  signatures des méthodes

- Implémentation invisible

  ATTRIBUTE AVS : STRING ;

  ATTRIBUTE nom : STRING ;

  ATTRIBUTE adresse : STRING ;

  ATTRIBUTE sal_mensuel : INT ;

  RELATIONSHIP service : Service
  INVERSE Service.personnes

  structure des données

Exemple d'encapsulation (2)

Implémentation invisible (suite) : code des méthodes

salaire ()

```
{ return sal_ mensuel }
```

newService (servoid: Service)

```
{ self.service := servoid }
```

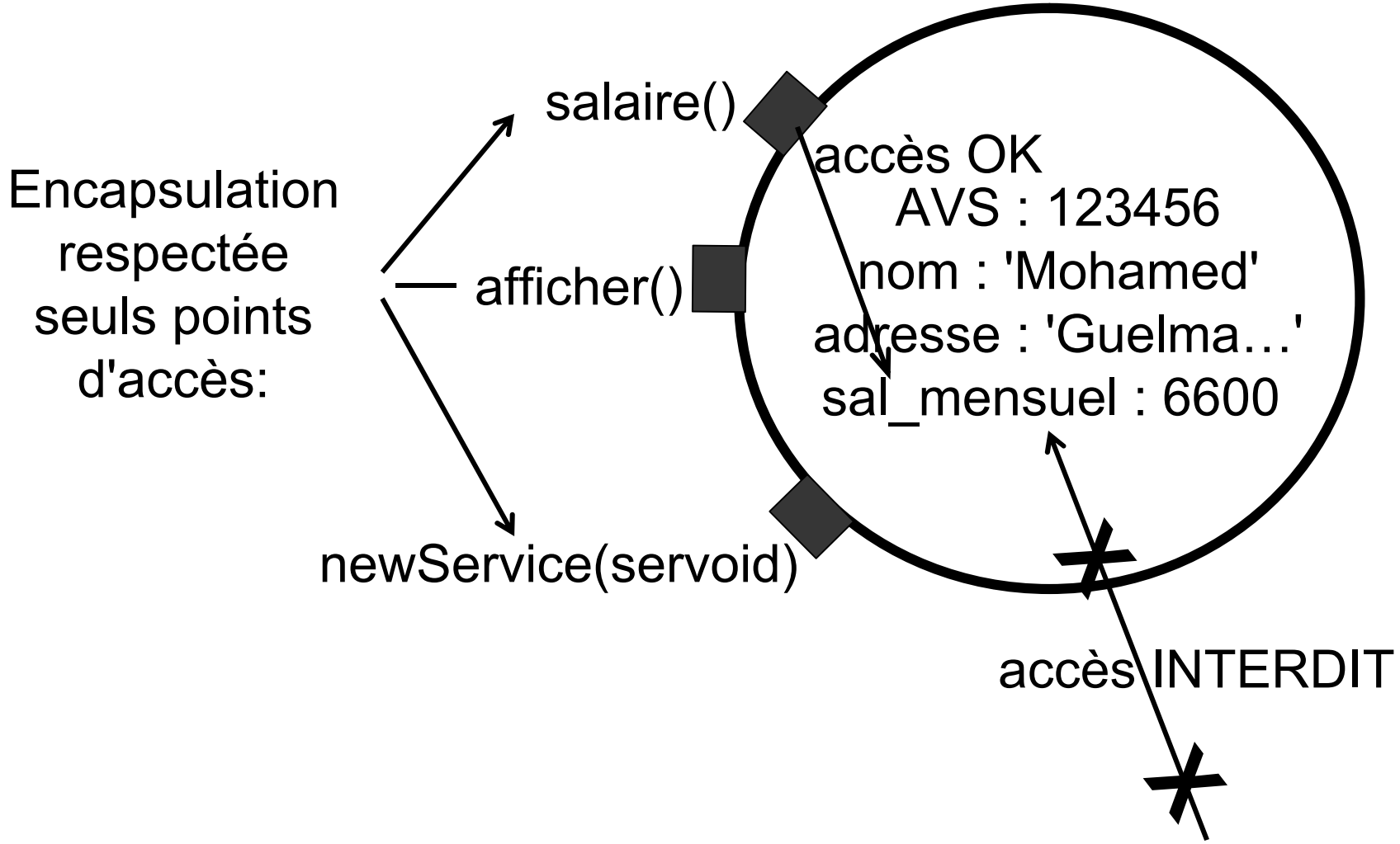
afficher ()

```
{
  PRINT('AVS:', self.AVS) ;  PRINT('nom:', self.nom) ;
  PRINT('adresse:', self.adresse) ;
  PRINT('salaire mensuel:', self.sal_mensuel) ;
}
```

Encapsulation : seul l'objet lui-même (c-à-d les instructions de ses méthodes) peut accéder à ses attributs

Exemple d'encapsulation (3)

- Un objet de la classe Personnel



Object Definition Language

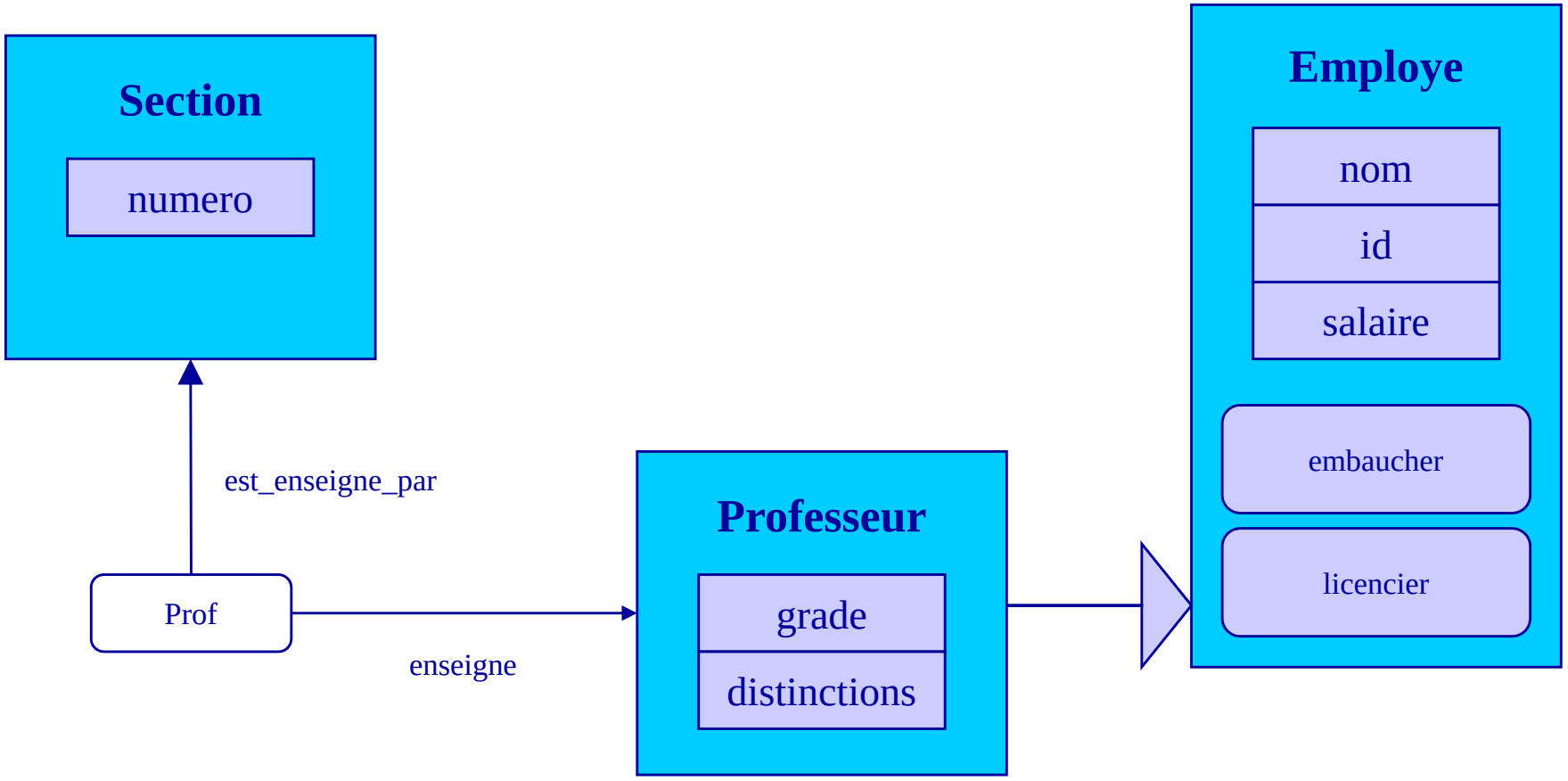
- ODL est un langage pour décrire le schéma des bases de données objet.

- ODL définit les types d'objet que l'on peut implémenter dans de nombreux langages de programmation:
  - ODL n'est pas lié, ni à la syntaxe, ni à la sémantique d'un langage de programmation.

- ODL est basé sur IDL, le *Interface Definition Langage* de l'OMG:
  - www.omg.org pour plus d'information.

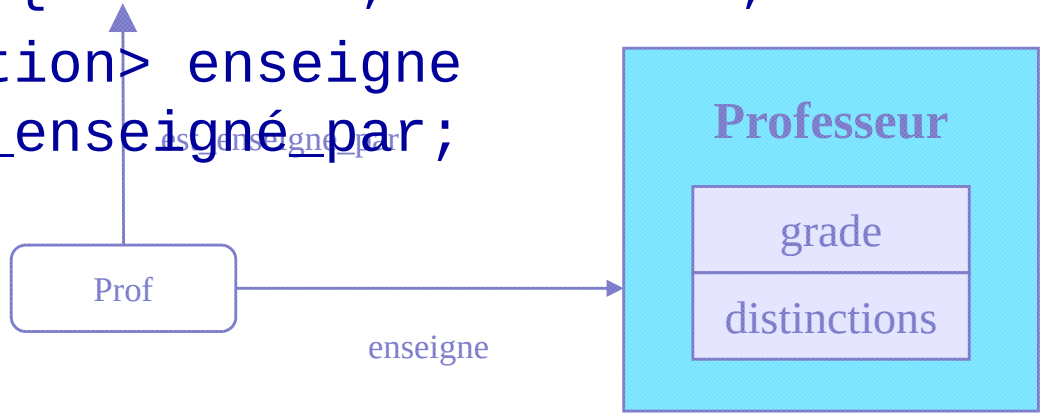
ODL - Déclaration d'un type (1)

Exemple simplifé



ODL - Déclaration d'un type (2)

```
interface Professeur:Employe{
   extent professeurs;
   attribute enum grade {titulaire, vacataire, assistant};
   relationship Set<Section> enseigne
   inverse Section::est_enseigné_par;
}
```



Object Query Language

- Permettre un accès facile à une base objet;
- offrir un accès non procédural pour permettre des optimisations automatiques (ordonnancement, index, ...);
- garder une syntaxe proche de SQL;
- rester conforme au modèle de l'ODMG;
- permettre de créer des résultats littéraux, objets, collections, ...;
- supporter des mises à jour limitées via les opérations sur objets, ce qui garantit le respect de l'encapsulation.

OQL et SQL3

Langage de requêtes de l’ODMG

L’objet-relationnel

Le langage OQL

- Permettre un accès facile à une base objet
  - via un langage interactif autonome
  - par intégration dans un des langages de l'ODMG (C++, Smalltalk, Java)
- Offrir un accès non procédural
  - permetttre des optimisations automatiques (ordonnancement, index,…)
  - garder une syntaxe proche de SQL
- Rester conforme au modèle de l'ODMG
  - application d'opérateurs aux collections extensions ou imbriquées
  - permettre de créer des résultats littéraux, objets, collections, …
- Permettre des mises à jour limitées via les méthodes

Caractéristiques

- Langage de requêtes fonctionnel
  - composition d'expressions
  - manipulation des types des langages de programmation objet
  - appel d'opérations avec mises à jour

- Fonctionnalités
  - select avec expressions de chemins
  - opérateurs ensemblistes : union, intersect, except
  - quantificateurs : for all, exists
  - agrégats : count, sum, min, max, avg
  - autres : order by, group by, define

Concepts nouveaux

- Expression de chemin mono-valuée
  - Séquence d'attributs ou associations mono-valués de la forme X1.X2…Xn telle que chaque Xi à l'exception du dernier contient une référence à un objet ou un littéral unique sur lequel le suivant s'applique.
  - Utilisable en place d'un attribut SQL
- Collection dépendante
  - Collection obtenue à partir d'un objet, parce qu'elle est imbriquée dans l'objet ou pointée par l'objet.
  - Utilisable dans le FROM

Forme des Requêtes

- Forme générale d'une requête

  Expressions fonctionnelles mixées avec un bloc select étendu

  Select [<type résultat>] (<expression> [, <expression>] ...)

  From x in <collection> [, y in <collection>]...

  Where <formule>

- Type résultat
  - automatiquement inféré par le SGBD
  - toute collection est possible (bag par défaut)
  - il est possible de créer des objets en résultat
- Syntaxe très libre, fort contrôle de type

Schéma

Class **Personne** {
String nom;
Date datenais;
Set <ref <Personne>> parents inverse enfants;
List <ref<Personne>> enfants inverse parents;
Ref <Appartement> habite inverse est-habite-par;
Int age();}
Class **Employe : Personne** { float salaire; }
Class **Appartement** {
int numero;
ref <Batiment> batiment inverse appartements;
ref <Personne> est-habite-par inverse habite; }
Class **Batiment** {
Adresse adresse;
List <ref>Appartement>> appartements inverse batiment;
}

Requêtes simples

Racines de persistance :

extensions des classes

set <Personne> lespersonnes;

set <Employe> lesemployes;

set <Appartement> lesappartements;

nommer un objet

martin est le nom d'un objet désignant l'employé qui s'appelle Martin.

Requêtes simples :

5 + 2	renvoie l'entier 7
martin	renvoie l'objet Employé de nom martin
martin.enfants	renvoie la liste des enfants de martin
martin.age	renvoie le résultat de la méthode age appliquée à martin

Sélection-Projection

Select … From… Where…

extraire d’une collection des éléments vérifiant une condition

Select e
From e in lesemployes
Where e.salaire > 2000

Select distinct e.salaire
From e in lesemployes
Where e.age < 30

Expression de chemin

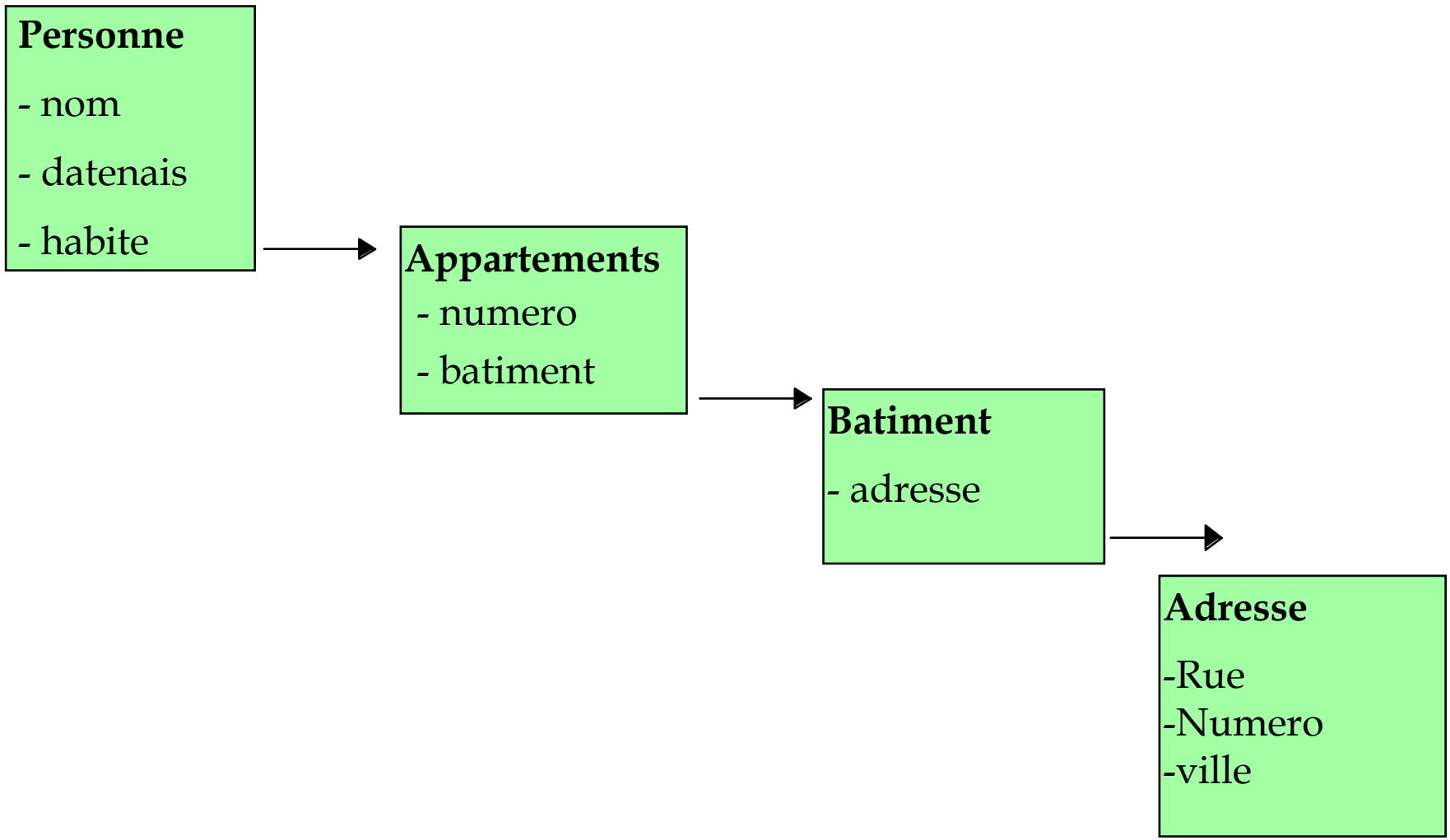
Un chemin permet de naviguer à travers les objets :

Ex. **p.habite.batiment.adresse.rue** (association 1-1)

Pour traverser des collections, on utilise une variable dans une expression select-from-where :

Ex. **select e.nom from e in p.enfants**

Navigation


Personne
- nom
- datenais
- habite
Appartements
- numero
- batiment
Batiment
- adresse
Adresse
-Rue
-Numero
-ville

Jointure

Relie plusieurs collections :

```
Select e.habite.batiment.adresse
From p in lespersonnes
        e in p.enfants
/* adresse des enfants de chaque personne */
Select p
from p in lespersonnes
         b in (select distinct a.batiment
                  from a in lesappartements)
Where p.nom = b.adresse.rue
/*Personnes qui habitent dans une rue qui porte leur nom*/
```

Construction des résultats

La structure du résultat est implicite.

Filtrer un ensemble (une liste) renvoie un ensemble (une liste).

On peut aussi construire des résultats à l'aide des constructeurs
struct, set, list, bag, array

Struct (nom : "Martin", salaire : 2000)

Select struct (personne : p.nom,
adresse : p.habite.batiment.adresse)

From p in lespersonnes

Résultats imbriqués

Imbrication des **select** au niveau **select**

```
Select struct (moi : p.nom,
                monadresse : p.habite.batiment.adresse,
                mesenfants : (select struct
                                (nom : e.nom,
                                adresse : e.habite.batiment.adresse)
                              from e in p.enfants))
From p in lespersonnes
```

Appel de méthodes

En OQL, on peut appeler des méthodes avec ou sans paramètres, dans la clause select ou dans la clause where.

```
Select max(select e.age
                    from e in p.enfants )
From p in lespersonnes
Where p.nom = “Paul “
```

Si la méthode a des paramètres, ils sont donnés entre parenthèses.

Création d’objets

Le résultat des requêtes est une valeur. On peut créer des objets en utilisant le nom de la classe. Certains attributs peuvent avoir une valeur par défaut.

Employe (nom : “Martin “ , salaire : 2000)

Employe (select struct (nom : enf.nom, salaire : 1500)

from p in lespersonnes

enf in p.enfants

where p.nom = “Martin “)

Opérateurs

Agrégats : **count, min, max, sum, avg**

ex: count(lesemployes)

Opérateurs ensemblistes : **union, intersect, except**

Define : permet de nommer le résultat d'une requête

ex : define <nom> as <requete>

Like : comparaison de chaînes

ex : like "Ma " renvoie tous les noms commençant par Ma

Quantificateurs

Quantificateur universel : **for all**

```
For all x in collection : predicat(x)
```

Ex : for all e in lesemployes : e.salaire > 2000

Renvoie true si tous les employes ont un salaire supérieur à 2000, false sinon.

Quantificateur existentiel : **exists**

```
Exists x in collection : predicat(x)
```

Ex : exists e in lesemployes : e.salaire > 5000 and e.age < 22

Opérateurs

Group by : regroupe les objets d'une collection selon les valeurs de certains attributs

Select e from e in lesemployes

Group by (bas : e.salaire <1000,

moyen : e.salaire >1000 and e.salaire < 3000

haut : e.salaire > 3000)

Renvoie struct (bas : set(emp), moyen : set(emp), haut : set(emp))

Order by : permet de trier

Select p from p in lespersonnes order by p.age, p.nom

Opérateurs de conversion

Element : extrait l'unique élément d'un ensemble

element (select e from e in lesemployes where e.nom = "Martin ")

Listtoset : transforme une liste en ensemble

Listtoset (list(1,2,3,2)) renvoie l'ensemble contenant 1, 2 et 3

Flatten : aplatit une collection imbriquée

Flatten (list (set 1,2,3), set(3,4,5,6), set(7))) renvoie l'ensemble contenant 1,2,3,4,5,6,7

L'objet-relationnel

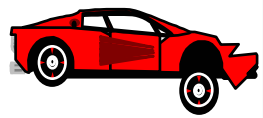
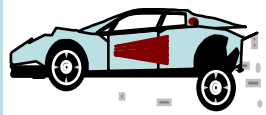
- Relationnel (tables, attributs, domaine, clé) + Objet (collections, identifiants, héritage, méthodes, types utilisateurs, polymorphisme)

- Extension du modèle relationnel
  - attributs multivalués : structure, liste, tableau, ensemble, ...
  - héritage sur relations et types
  - domaine type abstrait de données (structure cachée + méthodes)
  - identité d'objets
- Extension de SQL
  - définition des types complexes avec héritage
  - appels de méthodes en résultat et qualification
  - imbrication des appels de méthodes
  - surcharge d'opérateurs

Exemple de table et objet

Police	Nom	Adresse	Conducteurs	Accidents
24	Paul	Paris		

Conducteur	Age
Paul	45
Robert	17

Accident	Rapport	Photo
134		
219		
037		

Objet Police

Les concepts

- Extensibilité des types de données
  - Définition de types abstraits
  - Possibilité de types avec ou sans OID
- Support d'objets complexes
  - Constructeurs de types (tuples, set, list, …)
  - Utilisation de référence (OID)
- Héritage
  - Définition de sous-types
  - Définition de sous-tables

Le modèle SQL3 (ANSI99)

Extension objet du relationnel, inspiré de C++

- type = type abstrait de donnée avec fonctions
- fonction
  - associée à une base, une table ou un type
  - écrite en SQL, SQL3 PSM (Persistent Stored Module) ou langage externe (C, C++, Java, etc.)
- collection = constructeur de type
  - table, tuple, set, list, bag, array
- sous-typage et héritage
- objet = instance de type référencée par un OID (défaut)
- association = contrainte d'intégrité inter-classe

Les types abstraits

- CREATE TYPE <nom ADT> <corps de l'ADT>
- <corps de l'ADT>

  <subtype clause> ::= UNDER <supertype clause>

  possibilité d'héritage multiple avec résolution explicite

  <member list>

  <column definition> : attributs publics ou privés

  <function declaration> : opérations publiques

  <operator name list> : opérateurs surchargés

  <equals clause>, <less-than clause> : définition des ordres

  <cast clause> : fonction de conversion de types

Types abstraits

create type jour-ouvré (lun, mar, mer, jeu, ven, sam) ;

create type employé
    (nom char(10), datenais date, repos jour-ouvré) ;

create table employés of employé;

create table nouveaux-employés of employé ;

Les constructeurs de types

- Les types paramétrés
  - possibilité de types paramétrés (TEMPLATE)
  - généricité assurée par le compilateur ...
- Les constructeurs de base
  - collections SET(T), MULTISET(T), LIST(T)
  - CREATE TYPE person
    (nss INT, nom VARCHAR, prénoms LIST(varchar), tel SET(phone))
- Les références
  - possibilité de référencer un objet (avec OID)
  - CREATE TYPE car (number CHAR(9), color VARCHAR, owner REF(person))
- Les constructeurs additionnels
  - stack, queue, array, insertable array (exemple : texte)
  - non intégrés dans le langage mais peuvent être ajoutés

Les fonctions

- Définition des fonctions
  - [<function type>] : CONSTRUCTOR, ACTOR, DESTRUCTOR
  - FUNCTION <function name> <parameter list> RETURNS <function results> {<SQL procedure> | <file name>} END FUNCTION
- Peuvent être associées à une base, un type, une table, …
- Langage de programmation
  - SQL et SQL3 PSM, Langage externe

Fonctions

```
create type employé
    (nom char(10), datenais date, repos jour-ouvré,
     function age (e employé)
                    { < calcul de l'âge de e>};);

create table employés of employé ;

select nom
from employés e
where age(e) < 35;
```

Types structurés et fonctions

```
create type adr
   (rue char(20), codepost int, ville char(10),
   function dept (a adr) < return (nom-departement) >);

create table employés
   (nom char(10), adresse adr, repos jour-ouvré);

select nom, repos
from employés e
where dept (e.adresse) = "Ile de France";
```

Collections

```
create type adrs table of adr;
create type employé
   (nom char(10), adresses adrs, repos jour-ouvré);
create table employés of employé;

select nom                                        % noms des employés parisiens
from employés e
where "paris" in
            select ville from e.adresses;

create list nouveaux-employés of employé;

select nom                                        % nom du dernier employé arrivé
from nouveaux-employés e
where position (e) = 1;
```

Sous-typage et héritage

```
create type personne
    (nom char(10), adresse char(30), datenais date) ;

create type employé under personne
    (affectation char(10), repos jour-ouvré);

create type étudiant under personne
    (université char(20) no-étudiant integer) ;

create table personnes of personne
    (primary key (nom)) ;

create table employés under personnes of employé ;
```

Objets

```
create type employé (nom char(10), affectation site) ;
create type elts set of employé ;

create type site
    (nom char(10), budget integer, chef employé, emps elts);

create table employés of employé ;
create table sites of site ;

select nom                    % noms des employés affectés à Dupont
from employés
where affectation.chef.nom = "dupont" ;
```

Les tables

Caractéristiques

- une table peut posséder des attributs d'un type abstrait
- un tuple contient des références ou des valeurs complexes
- un attribut peut être de type référence (REF <type> ou avec OID)

Possibilité d'utiliser un type prédéfini

- CREATE TABLE cars OF car ;

Possibilité de définir un nouveau type

- Le type est celui des tuples de la table
- CREATE TABLE Constructors OF NEW TYPE Constructor (name VARCHAR, total MONEY) ;

Possibilité de définir des sous-tables

- CREATE TABLE FrenchConstructors UNDER Constructors(taxe MONEY)

L'appel de fonctions et opérateurs

- Appel de fonctions

```
SELECT r.name
FROM emp j, emp r
WHERE j.name = 'Joe' and  distance(j.location,r.location) < 1 ;
```

- Appel d'opérateurs

```
SELECT r.name
FROM emp, emp r
WHERE emp.name = 'Joe' and
    contained(r.location, circle(emp.location,1)) ;
```

Le parcours de référence

- Possibilité d'appliquer les fonctions Ref et DeRef (implicite)
  - CREATE TABLE cars OF TYPE car
  - SELECT c.Owner.name FROM cars c
    WHERE color = 'red'
- Possibilité de cascader la notation pointée
  - SELECT dname FROM dept WHERE 1985 IN auto.years
- Généralisation possible aux chemins multiples
  - SELECT dname FROM dept
    WHERE autos.(year=1985 and name = 'Ford')
- Toute collection peut être utilisée en place d'une table

Conclusion

SQL3, standard en évolution, proposé par tous les grands constructeurs (Oracle, Sybase, IBM, etc.), sous une forme limitée.

Compatibilité avec le relationnel.

ODMG et SQL3 : deux propositions complémentaires