

BDR  
Master d'Informatique (SAR)

Cours 3- Objet et objet-relationnel

Stéphane Gañcarski  
Stephane.Gancarski@lip6.fr

1

Objet et Objet-relationnel

- Introduction
- Bases de données objet
- Le modèle objet-relationnel

2

Insuffisances du modèle relationnel

- Opérations séparées des données
  - procédures stockées non intégrées dans le modèle
- Limites du modèle de données
  - 1ère forme normale de Codd
  - inadapté aux objets complexes (documents structurés)
  - introduction de BLOB
- Dysfonctionnement des langages (impédance de types)
- Mauvais support des applications non standards
  - CAO, CFAO
  - BD Géographiques
  - BD techniques

3

L'apport des modèles objets

- Identité d'objets
  - introduction de pointeurs invariants
  - possibilité de chaînage
- Encapsulation des données
  - possibilité d'isoler les données par des opérations
  - facilite l'évolution des structures de données
- Héritage d'opérations et de structures
  - facilite la réutilisation des types de données
  - permet l'adaptation à son application
- Possibilité d'opérations abstraites (polymorphisme)
  - simplifie la vie du développeur

4

## Définitions

BDO = base de données dont les éléments sont des objets  
(avec identité et opérations)  
schéma BDO = graphe de classes des objets de la base

SGBDO = SGBD supportant

- langages objet (C++, Smalltalk, Java, etc.)
- objets structurés et non structurés
- objets volumineux (multimédias)
- objets complexes (avec associations inter-objets)
- composants et appel d'opérations
- héritage et surcharge d'opération

5

## Objet

Un objet est constitué

- de données
- d'opérations applicables à ces données

On distingue l'*interface* (description des opérations applicables à l'objet) de l'*implémentation* (structure de données physiques et procédures qui réalisent les opérations)

L'utilisateur ne voit que l'interface. Les données sont encapsulées.

Create type

Create type body

6

## Objet complexe

Les objets complexes sont construits à partir d'objets atomiques et de constructeurs.

Les objets atomiques sont les entiers, les réels, les booléens, les chaînes de caractères, les images, etc.

Les constructeurs sont le n-uplet, l'ensemble, la liste, et le tableau.

L'utilisation des constructeurs est indépendante du type des objets (orthogonalité du système de types)

7

## Identité d'objet

Chaque objet a une identité indépendante de sa valeur.

Identité et égalité sont deux concepts différents.

L'identité d'objet permet :

- Le partage d'objet
- La représentation d'objets cycliques (graphe de composition)
- Le maintien automatique de la contrainte référentielle
- La mise à jour des valeurs sans changer l'identité
- Plusieurs niveaux de comparaison (égalité superficielle et égalité profonde)

8

## Types et classes

Un type/classe décrit les propriétés communes d'un ensemble d'objets :

- propriétés structurelles (données)
- propriétés de comportement (méthodes)

Un type/classe a deux parties : spécification (méthodes) et implémentation (structure de données et implémentation des méthodes)

Les types ou les classes permettent une abstraction des informations et de leur représentation. Ce sont les concepts utilisés pour décrire le schéma de la base.

C'est aussi un mécanisme de vérification de la justesse des programmes.

9

## Héritage et sous-typage

On peut factoriser des classes ayant des propriétés communes (structure et /ou méthodes). Une sous-classe hérite des propriétés (données et méthodes) de sa super-classe. Elle est spécialisée en ajoutant des attributs propres et des méthodes propres.

Ex:

classe Personne (nom : string, datenais : Date)

classe Etudiant inherits Personne (cursus : string)

10

## Héritage

L'héritage permet d'éviter la répétition de code en facilitant la ré-utilisation.

Il rend le code plus simple en plaçant chaque procédure à son niveau d'abstraction optimal.

Il fournit une description compacte et bien structurée du schéma.

Un schéma de base de données objet : ensemble de classes organisées en hiérarchie.

- Héritage simple : arbre
- Héritage multiple : DAG

Ne pas confondre avec  
graphe de composition

11

## Redéfinition et Surcharge

Redéfinition :

Une méthode spécifiée dans une super-classe peut être redéfinie, (avec le même nom) avec un code différent au niveau d'une sous-classe.

Surcharge :

Possibilité de définir plusieurs codes pour une même opération d'une classe. Le code approprié est sélectionné selon le type des paramètres fournis lors d'un appel.

Ex. Un objet peut avoir plusieurs méthodes *afficher*, selon qu'il s'agit d'afficher un formulaire, une carte, une image, etc.

12

## Persistence

Un objet persistant est un objet stocké dans la base, dont la durée de vie est supérieure au programme qui le crée.

Une donnée est rendu persistante par une commande explicite (LP) ou par un mode automatique (BD):

attachement à une racine de persistance

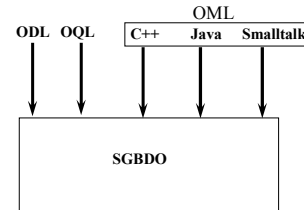
attachement à un type de données

Les noms du schéma servent de points d'entrée à la base de données (les relations en relationnel, les noms de classe, ou des objets nommés en BDOO)

13

## ODMG

- Définit les standards pour les BDO (modèle, ODL, OQL)
- Adaptation du modèle objet de l'OMG
- Interfaces d'accès à un SGBDO avec les LPO



14

## Le Modèle

Extension du modèle de l'OMG

- l'OMG a proposé un modèle standard pour les objets
- le modèle est supporté par le langage IDL (def. interface)
- les BD objets nécessitent des adaptations/extensions
  - instances de classes
  - collections
  - associations
  - persistance
  - transactions

Un candidat pour un profil BD de l'OMG

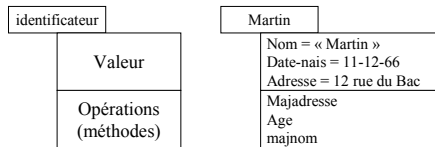
15

## Les Objets : Instances

- Identifiés par des OID :
  - gérés par le SGBD OO pour distinguer les objets
  - permettent de retrouver les objets, reste invariant
  - peuvent être nommés par les utilisateurs
- Persistants ou temporaires :
  - les objets persistants sont les objets BD, les autres restant en mémoire
- Peuvent être simples ou composés :
  - atomiques, collections, structurés
- Héritage d'un type racine :
  - propriétés communes : création, verrouillage, comparaison, copie, suppression

16

## Exemple



17

## Les objets collections

Cf. tableau associatif PHP

Support de collections homogènes :

- Set<t>, Bag<t>, List <t>, Array<t>, Dictionary<t,v>
  - héritent d'une interface commune collection :
    - Cardinalité, existence d'un ordre, tester si la collection est vide ou non, si un élément appartient à la collection, si les doublés sont autorisés.
    - Insertion, suppression d'un élément
    - Création d'un itérateur (unidirectionnel ou bi-directionnel) et des opérations associées.
  - Chaque collection a une interface spécifique
- Hiérarchie de Types

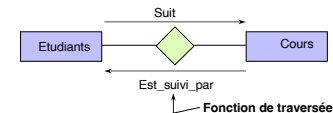
18

## Les Attributs

- Une propriété permettant de mémoriser un (ou plusieurs) littéral/objet
- Peut être vue comme deux fonctions :
  - Set\_value
  - Get\_value
- Propriétés:
  - son nom
  - type de ses valeurs légales
- Un attribut déclaré au niveau interface n'est pas forcément implémenté (peut être calculé).

19

## Les Associations (Relationships)



- Associations binaires, bi-directionnelles de cardinalité (1:1), (1:N), (N:M).
- Opérations:
  - Add\_member, Remove\_member
  - Traverse, Create\_iterator\_for

20

## Exemple

- Interface étudiants { ... Relationship list<cours> suivre  
inverse cours::est\_suivi\_par; };
- Interface cours { ... Relationship set<étudiants>  
est\_suivi\_par inverse étudiants: :suivre; };
- Le SGBDO est responsable du maintien de l'intégrité (« cascade »)
  - Référence dans les deux sens si inverse déclaré
  - Ce n'est pas le cas pour un attribut valué par un objet  
attribut suivre list<cours>  
pas de chemin inverse, pas d'intégrité référentielle

21

## Les Opérations

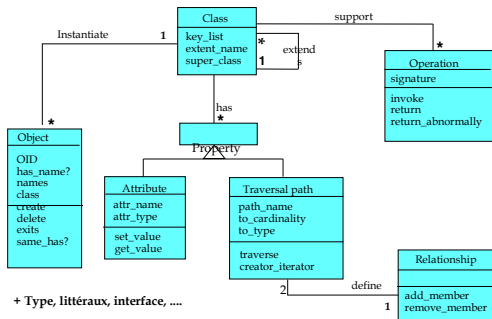
Représentent le comportement du type

Propriétés:

- nom de l'opération
- nom et type des arguments (in)
- nom et type des paramètres retournés (out)
- nom des conditions d'erreurs (raises)

22

## Méta-modèle du modèle ODMG



24

## 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
  - permettre 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

25

## 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

26

## 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;  
    }  
}
```

27

## Exemples de requêtes

**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.

**Select e From e in lesemployes**

**Where e.salaire > 2000**

*/\* Employés dont le salaire est supérieur à 2000 \*/*

**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 \*/*







28

## 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

29

## Exemple de table et objet

Police	Nom	Adresse	Conducteurs		Accidents		
24	Paul	Paris	Conducteur	Age	Acciden	Rapport	Photo
			Paul	45	134		
			Robert	17	219		
					037		

Objet Police

30

## 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 (ref ≈ OID)
- Héritage
  - Définition de sous-types
  - Définition de sous-tables

31

## 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 PL/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

32



## Types (SQL3 Oracle)

- Types atomiques
  - Varchar2(<longueur>)
  - Number(<longueur>)
  - Date
- Types objet

```
Create type <nom-type> as Object (  
    (<nom-attribut> [ref] <type>, )+  
    (<declaration-methode>,)*)  
);
```
- Types ensemblistes :
  - Table (ensemble avec doublons)
  - Varray (collection ordonnée avec doublons)

```
Create type <nom-type> as  
    (Table | Varray(<longueur>) ) of <type>;
```

33

## Types abstraits

```
create type numsecu varchar2(15) ;  
  
create type employe as Object (  
    nom varchar2(10),  
    nss numsecu,  
    datenais Date) ;  
  
create type Employes as Table of employe;  
  
create type Nouveaux-employes as Varray (10) of  
    employe ;
```

34

## Méthodes

Fonctions ou procédures associées à un type d'objet.  
Modélisent le comportement d'un objet  
Ecrites en PL/SQL ou JAVA, et sont stockées dans la base.  
Déclaration :

```
Member function <nom-fonction>  
    [(<nom-para> in <type>, ...)] return <type-resultat>  
Member procedure <nom-proc>  
    [(<nom-para> in <type>, ...)]
```

Le corps de la méthode est défini dans la classe du receveur.

```
Create type body <type-objet> as  
    <declaration-methode> is  
    <declaration var-locales>  
begin <corps de la methode> end;
```

35

## Exemple

```
create type employe as Object (  
    nom varchar2(10),  
    nss numsecu,  
    datenais Date,  
    member function age return Number) ;  
  
create type body employe as  
    member function age return Number is  
    begin  
        return sysdate - datenais;  
    end age;  
end;
```

36

## Sous-typage et héritage

```
create type personne
(nom varchar2(10), adresse varchar2(30), datenais date) ;

create type employé under personne
(affectation varchar2(10), repos jour-ouvré);

create type étudiant under personne
(université varchar2(20) no-étudiant integer) ;

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

create table employés under personnes of employé ;
```

37

## Langage de requête

Standard SQL étendu à l'objet-relationnel :

```
SELECT [distinct] ... FROM ... [WHERE ...]
```

La clause SELECT peut contenir, une variable, un attribut, un nom de fonction, un chemin (notation pointée).

Les clauses FROM et WHERE peuvent contenir des requêtes imbriquées.

38

## Exemples

```
create type adr as table of adr;
create type employé as object
(nom varchar2(10), adresses adr);
create table employés of employé;

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

39

## Exemples

```
create table LesAdresses of adr;
create type employé as object
(nom varchar2(10), adresse ref adr);
create table employés of employé;

INSERT INTO employés
VALUES employé('Peuhplu Jean',
adr((SELECT REF(i) FROM LesAdresses i
      WHERE i.rue='du Soleil'
      AND i.numero=3
      AND i.situé.codePostal='75020'))),
```

40

## Objets

```
create type employé as object
(nom varchar2 (10), affectation site) ;
create type elts as table of employé ;

create type site as object
(nom varchar2(10), budget number, chef ref
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" ;
```

41

## Conclusion

- Deux approches différentes :
  - les SGBD objet mettent en commun les fonctionnalités des bases de données et celles des langages de programmation (objet),
  - les SGBD objet-relationnel complètent le modèle relationnel avec un système de types plus riche
- Et complémentaires
  - Les SGBD objet sont mieux adaptés aux applications où l'objet a une position centrale : applications manipulant des objets structurés et non structurés, ayant des associations complexes, faisant appel à des sources de données hétérogènes, etc.
  - Les SGBD objet-relationnel sont optimisés pour gérer des applications manipulant de très grandes collections de données, nécessitant de nombreuses transactions (relativement courtes), ayant besoin des fonctionnalités traditionnelles des SGBDR en général.

42