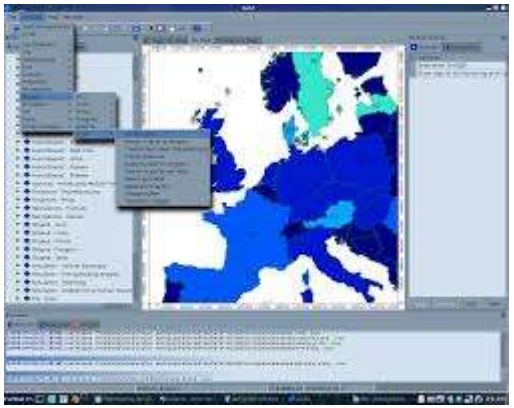
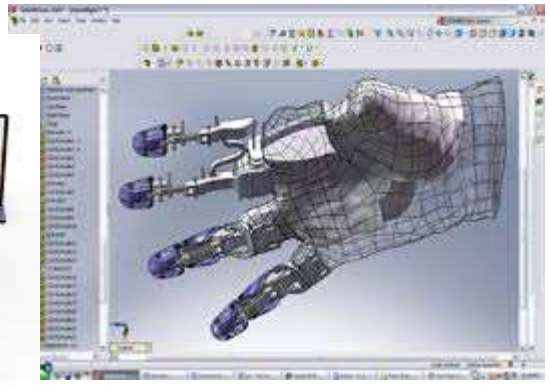
Chapitre 3

Bases de Données Objet

Nouvelles applications

- De nouvelles applications avec de nouveaux besoins se multiplient
- Différentes des applications de gestion classiques

Nouvelles applications (2)

- Exemple
  - ✓ Conception / Production assistée par ordinateur
  - ✓ Génie logiciel
  - ✓ Bureautique
  - ✓ Recherche d'informations sur le Web
  - ✓ Systèmes d'informations géographiques
  - ✓ etc

- Ces applications ont révélé les limites du modèle relationnel

Limites du modèle relationnel

- Modèle de données trop simple
  - ✓ Ne permet pas de représenter facilement les entités du monde réel qui sont souvent complexes
  - ✓ Pas d'attribut complexe, ni multivalué
  - ✓ Un seul type de lien (clé externe)

- Incompatibilité des LMD relationnels et des langages de programmation
  - ✓ Les LMD sont déclaratifs et fournissent en résultat un ensemble de tuples
  - ✓ Les langages de programmation sont impératifs et travaillent sur un élément à la fois

Limites du modèle relationnel (2)

- ✓ Les types de données manipulés par les langages de programmation sont plus variés et peuvent être plus complexes que les domaines de valeurs des LMD relationnels
- ✓ Le modèle relationnel manipule uniquement les données alphanumériques

- Mécanisme de transactions inadapté aux nouvelles applications

Les Bases de Données Orientées Objets

- Cherchent à répondre à ces nouveaux besoins
- Elle sont nées de la convergence de deux domaines
  - ✓ Les bases de données
  - ✓ Les langages de programmation orientés objets, tels que Java, C++, Smalltalk, Eiffel, …

Modèle à Objets

- Extension de concepts de langages O.O aux bases de données ou la persistance des données est primordiale
- Concepts clés :
  - ✓ Types
  - ✓ Classes et objets
  - ✓ Identité des objets
  - ✓ Héritage
  - ✓ Liens de compositions

Objectif des BD OO

- Rendre les objets persistants après la fin de l'exécution du programme
- Partager les objets entre plusieurs programmes qui s'exécutent en parallèle
- Partager entre plusieurs instances d’applications
- Lire des objets de manière transparente (persistance transitive)
- Effectuer des transactions sur les objets afin de préserver la cohérence des données

Exemple de SGBDs object

- Il existe plusieurs systèmes de gestion de base de données objet
- Exemple:
  - ✓ ObjectStore
  - ✓ ObjectDB
  - ✓ Db4o

ODMG

- ODMG: (*Object Database Management Group)*
- www.odmg.org
- Groupe de normalisation des SGBD OO
- Fondé à l'initiative de SUN en 1991 par cinq constructeurs de SGBD objet : O2 Technology, Objectivity, Object Design, Ontos et Versant
- A regroupé de nombreux autres vendeurs de SGBO OO:
  - ✓ Poet
  - ✓ Ardent
  - ✓ GemStone
  - ✓ et des constructeurs, des utilisateurs et des chercheurs, etc

JDO

- JDO (*Java Data Objects)*
- *S*tandard pour la définition de SGBD orienté objet

Caractéristiques des SGBD OO

- Tout objet peut être rendu persistent
- La structure des objets peut être arbitrairement complexe
- Supporte l'héritage
- Supporte les relations entre les objets (persistance transitive)
- Chaque objet est identifié de manière unique par un identificateur d'objet
- L'identificateur d'objet est une valeur interne au SGBD OO, elle n'est pas accessible a l'utilisateur de la BD

Persistance

- Deux types d'objets: transitoire et persistant
- Objet transitoire :
  - ✓ Un objet qui n'existe qu'en mémoire vive
  - ✓ Il disparait à la fin de l'exécution du programme
- Objet persistant :
  - ✓ Un objet qui est stocké sur disque et qui peut être chargé en mémoire vive
  - ✓ Créé durant l'exécution d'un programme et sauvegardé avant la fin de l'exécution
  - ✓ Pour être stocké, l'objet doit avoir un nom persistant, ou bien il doit être accessible a partir d'un objet ayant un nom persistant

Persistance transitive

- Permet de stocker les objets persistants
- Lorsqu'un objet persistant est sauvegardé, tous les objets persistants qu'il référence sont aussi sauvegardé
- Si la persistance transitive n'est pas supporté, il faut charger ou sauvegarder manuellement toutes les données

Modélisation

Modélisation des BD OO

- Les bases de données orientées objets se caractérisent par une diversité de modèles
  - ✓ Il n'existe pas un seul modèle de données orienté objets, mais plusieurs
  - ✓ Norme ODMG, mais elle n’est pas suivie par de nombreux SGBDO
- Nous utilisons dans ce cours une syntaxe tirée de celle de l’ODMG

Structure

Structure des données - Objet

- Concept principale : Objet
  - ✓ La structure de données des modèles OO se base sur un seul concept principal: l'objet
- Les objets sont
  - ✓ décrits par des attributs
  - ✓ et sont regroupé en classes
- Chaque objet a une identité permettant de le distinguer des autres objets
  - ✓ L'identité de l'objet est permanente
  - ✓ OID (Object Identity)

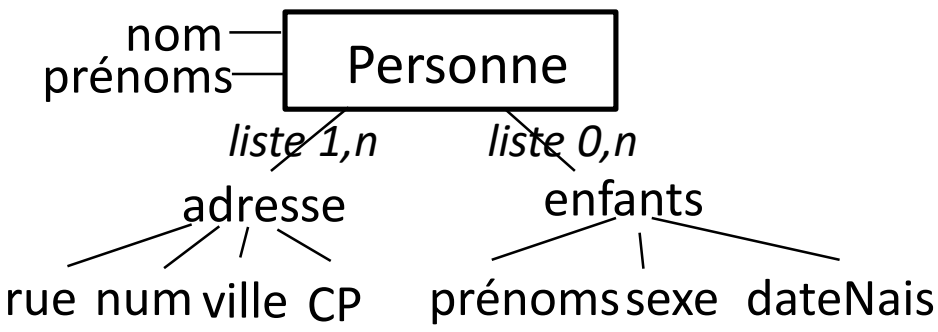
Structure des données - Liens

- Il existe deux types de liens entre les objets : liens de composition et liens de généralisation/spécialisation
- Liens de composition entre les objets
  - ✓ Un objet est composé d'un ensemble d'objets
  - ✓ Décrits par des attributs-référence, qui ont pour domaine une classe d'objets
- Liens de généralisation / spécialisation:
  - ✓ Les liens de type *est_un* (*IS-A*)
  - ✓ Mécanisme d'héritage des propriétés d'une classe génériques par une classe spécialisée (détaillées dans le chapitre précédent)

Structure complexe

- Les SGBDO permettent de définir des structures quelconques et qui peuvent être complexes
  - ✓ Pour pouvoir représenter le monde réel
  - ✓ Les attributs peuvent être complexes et multivalués

- Exemple: Représenter une personne
  - ✓ Monde réel : Personne: Nom, Prénoms, adresse (rue, num, ville, CP), enfants (prénoms, sexe, dateNais)
  - ✓ Relationnel: des relations et des tuples
  - ✓ OO: Un seul objet



Structure

- Les structures sont utilisées pour
  - ✓ définir la structure des objets (la structure de leurs classes)
  - ✓ étendre l'ensemble des types (appelés aussi domaines) prédéfinis qui existent dans le SGBDO (Tles que STRING, INT, DATE, MONEY…) en définissant des domaines spécifiques aux application des utilisateurs

- Les structures sont définies en employant des constructeurs
  - ✓ Constructeur de structure complexe
  - ✓ Constructeurs de collection

Constructeur de structure complexe

- STRUCT
- L'équivalent d'un attribut complexe
- Permet de créer une structure composée d'une suite d'attributs:
  - ✓ Syntaxe: STRUCT {nom_att_1: dom_1; nom_att_2: dom_2; ...}
  - ✓ où dom_i est le domaine de l'attribut de nom nom_att_i
- Le domaine dom_i peut être
  - ✓ domaine prédéfini (STRING, FLOAT, INT, DATE...)
  - ✓ un type défini par un constructeur
  - ✓ Le nom d'une classe

Constructeurs de collection

- L'équivalent des attributs multivalués
- SET : créer un type ensemble d'éléments
- LIST: liste
- BAG: multi-ensemble (ensemble pouvant contenir des doubles)
- ARRAY: tableau

- Question: Quelle est la différence entre la structure SET et List ? (En Java par exemple)

Types définis par l'application

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

Exemple – V1

- Une classe avec des attributs complexes et multivalués

```
CLASS Etudiant                          /* déclaration d'une classe d'objets */
  { num : INT ;
    nom : STRING ;
    prénoms : LIST STRING ;             /* attribut multivalué de type liste */
    dateNais : DATE ;
    adresse : STRUCT {  num: INT ;                  /* attribut complexe */
                        rue : STRING ;
                        ville : STRING ;
                        CP: INT} ;
    coursSuivis : SET STRUCT { nomCours : STRING ;   /* attribut complexe
                               note : FLOAT}      }          et multivalué */
```

Exemple – V2 (2)

- Même exemple, mais avec déclaration d'un nouveau domaine

```
TYPEDEF T_Adresse                          /* déclaration d'un domaine
    STRUCT {  num : INT ;                     de type complexe */
              rue : STRING ;
              ville : STRING ;
               CP:   INT     }
```

Exemple – V2 (3)

CLASS Etudiant

{ num : INT ;

nom : STRING ;

prénoms : **LIST** STRING ;

datNais : **DATE** ;

adresse : **T_Adresse**; /* attribut de domaine complexe */

coursSuivis : **SET STRUCT** { nom-cours : STRING ; note : FLOAT} **}**

- Question: Quelle est la différence entre ces deux versions de l'exemple ?

Exemple (4)

- La structure de la classe Etudiant est exactement la même
- Intérêt de déclarer un nouveau domaine T_Adresse: **Réutilisation**
  - ✓ Il peut être réutilisé comme domaine dans d'autres classes
  - ✓ et lors de la définition d'autres domaines

Identité d'objet

Identité d'objet

- Objectif: Pouvoir identifier tout objet indépendamment de sa valeur et de son adresse (MC ou Disque)
- Cela implique
  - ✓ Pouvoir gérer des objets distincts ayant les mêmes valeurs
  - ✓ Pouvoir gérer les changement de valeurs et des déplacement internes
- Chaque objet possède un identifiant qui doit être
  - ✓ Permanent: existe pendant toute la durée de vie de l'objet
  - ✓ immuable: ne change pas durant la vie de l'objet
  - ✓ Unique (dans la base et dans le temps): deux objets distincts de la même base, même s'ils n'existent pas en même temps n'auront jamais la même identité

Identifiant d'objet

- OID
- La référence de l'objet
- Est géré par le SGBDO
  - ✓ Dès la création d'un objet, un identifiant système lui est attaché
  - ✓ La valeur de l'identifiant n'est ni affichable, ni imprimable, ni modifiable

- Objet : (oid, valeur)

Test d'égalité

- L'égalité de deux objets peut se mesurer à plusieurs niveaux:
  - ✓ Identité d'objets, Egalité de valeur, Egalité de valeur après fermeture

- Identité d'objets
  - ✓ Noté: ==
  - ✓ Signifie: est-ce le même objet (le même OID) ?

- Egalité de valeur
  - ✓ Noté: =
  - ✓ Les objets ont-ils la même valeur ?
  - ✓ C.-à-d., tous leurs attributs ont la même valeur, que ce soient des attributs-valeur ou des attributs-référence

Test d'égalité (2)

- Egalité de valeur après fermeture (Noté: $=_f$)
  - ✓ Les objets ont-ils la même valeur après fermeture ?
  - ✓ C.-à-d., ont des graphes de composition isomorphes
  - ✓ Leurs attributs-référence constituent des graphes équivalents
  - ✓ Et les attributs-valeur correspondants, à tous les niveaux, sont égaux.

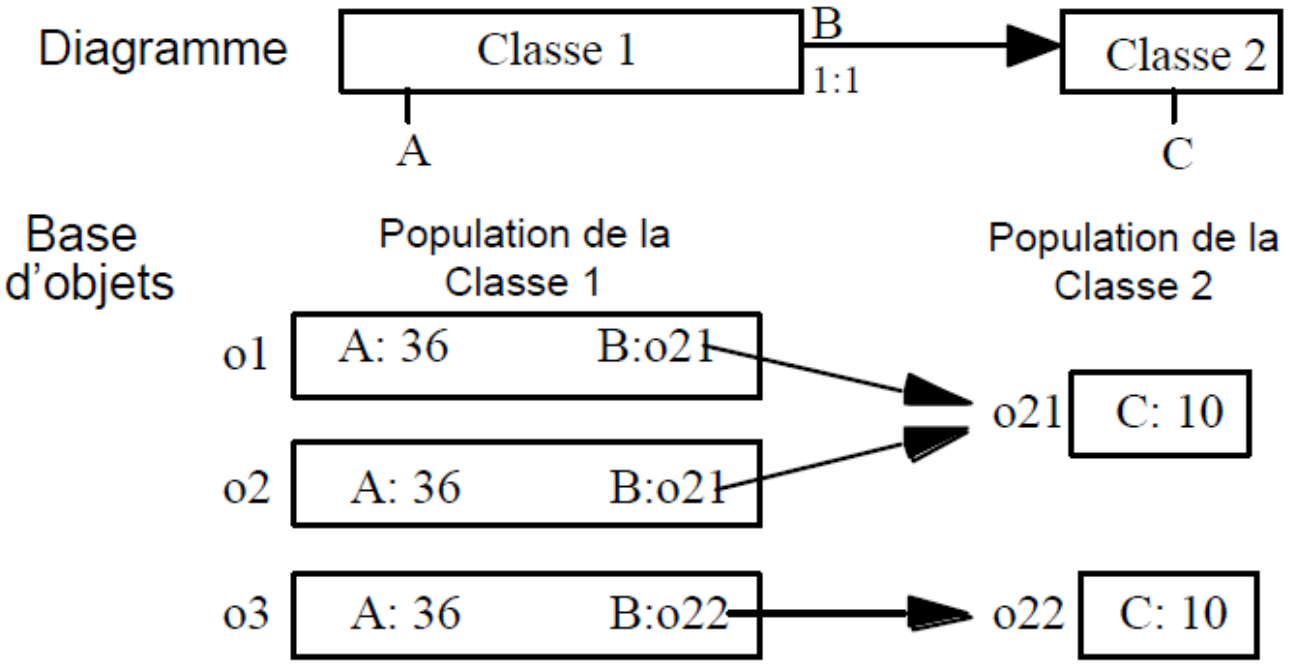
- Pour deux objets, o1 et o2, de la même classe, on a les propriétés suivantes:
  - ✓ $o1 == o2 \Rightarrow o1 = o2$
  - ✓ $o1 = o2 \Rightarrow o1 =_f o2$

Comparaison

- Effectué selon le même principe que dans les langages de programmation
  - ✓ chaque élément d'un programme possède une identité (nom de la variable ou adresse physique)

Test d’égalité - Exemple

- Nous avons 2 classes d'objets Classe 1 et Classe 2
  - ✓ La classe 1 possède un attribut A de domaine entier et un attribut référence B qui contient l'oid d'un objet de la classe 2
  - ✓ La classe 2 possède un attribut C de domaine entier



Test d'égalité – Exemple (2)

- Question: Compléter te tableau suivant:

	o1.B == o2.B	o1 == o2	o1 = o3	o1 = o2	o21 == o22	o21 = o22	o1 $=_f$ o3
Vrai							
Faux							

Test d'égalité – Exemple (2)

- Réponse

	o1.B == o2.B	o1 == o2	o1 = o3	o1 = o2	o21 == o22	o21 = o22	o1 $=_f$ o3
Vrai	✔			✔		✔	✔
Faux		✔	✔		✔		