Chapitre VIII

Les bases de données

Orientées Objet

Motivation

Le modèle relationnel connaît un très grand succès et s'avère très adéquat pour les applications traditionnelles des bases de données (gestion).

Il est beaucoup moins adapté aux nouvelles applications plus complexes telles que :

- CAD/CAM (Computer-Aided Design/Computer-Aided Manufacturing)
- BD d'images et de graphiques
- BD géographiques (GIS : Geographic Information Systems)
- BD multimédia (son, image, texte, etc. combinés)

Motivation (suite)

Ces nouvelles applications ont des caractéristiques différentes des applications traditionnelles de gestion et elles introduisent des besoins nouveaux, notamment :

- des structures d'objets plus complexes,
- des transactions de durée plus longue,
- de nouveaux types de données pour le stockage d'images ou de gros documents de texte,
- la possibilité de définir des opérations non standards qui sont spécifiques aux applications,

Les BD orientées objet constituent une tentative de réponse à (certains de) ces besoins nouveaux.

Motivation (suite 2)

Une caractéristique importante des BD orientées objet est qu'elles donnent au concepteur de la BD la capacité de spécifier

– non seulement la structure d'objets complexes,

– mais aussi les opérations à appliquer à ces objets.

On recense actuellement plusieurs prototypes expérimentaux et quelques produits commerciaux de BD orientées objet

Ces systèmes reprennent en général les concepts adoptés dans les langages de programmation orientés objet, avec les spécificités des systèmes de bases de données (persistance des données, transactions, etc.).

Insuffisance des bases de données relationnelles : un exemple

On désire conserver dans une base de donnée géographique la description d'un réseau routier.

- Pour définir le réseau routier, on part de *points d'intersection* qui sont les points où plusieurs routes se croisent ou qui correspondent à un changement de caractéristiques d'une route. Un point d'intersection est caractérisé par ses coordonnées géographiques (latitude et longitude).
- On considère alors des *segments de route* qui sont des tronçons de route situés entre deux points d'intersection. Outre ses points d'origine et de destination, une information de catégorie (deux bandes, quatre bandes, ...) caractérise chaque segment.
- Une route est désignée par un identifiant (N4, E411, A602, ...) et est décrite par un ensemble de segments. De plus pour chaque route on conserve la désignation de l'autorité (région, commune, ...) qui la gère.

Insuffisance des bases de données relationnelles : un exemple (suite)

- Cette information ne peut être représentée naturellement dans le modèle relationnel, car on ne peut y définir de relations entre éléments qui sont des tuples ou des ensembles.
- On est donc amené à introduire de relations auxiliaires et des identifiants d'objets que l'on doit gérer explicitement.
- Un schéma possible serait :

  POINTS(ID_P, LATITUDE, LONGITUDE)

  SEGMENTS(ID_SEG, ID_P1, ID_P2, CATEGORIE)

  ROUTES(ID_R, ID_LISTE_SEG, AUTORITE)

  LISTES_SEGMENTS(ID_LISTE_SEG, ID_SEG)

- Il faut noter que l'usage des identifiants permet par exemple d'avoir deux segments de route entre les mêmes points, ou de représenter sans ambiguïté qu'un même segment fait partie de plusieurs routes.

Un contenu possible de la base de données est alors

POINTS	*ID_P*	*LATITUDE*	*LONGITUDE*
	P#1	*lat1*	*long1*
	P#2	*lat2*	*long2*
	P#3	*lat3*	*long3*
	⋮	⋮	⋮

SEGMENTS	*ID_SEG*	*ID_P1*	*ID_P2*	*CATEGORIE*
	SEG#1	*P#11*	*P#12*	*cat1*
	SEG#2	*P#21*	*P#22*	*cat2*
	SEG#3	*P#31*	*P#32*	*cat3*
	⋮	⋮	⋮	⋮

ROUTES	*ID_R*	*ID_LISTE_SEG*	*AUTORITE*
	R#1	*LS#1*	*autorite1*
	R#2	*LS#2*	*autorite2*
	R#3	*LS#3*	*autorite3*
	⋮	⋮	⋮

LISTES_SEGMENTS	*ID_LISTE_SEG*	*ID_SEG*
	LS#1	*SEG#1*
	LS#1	*SEG#2*
	LS#1	*SEG#3*
	LS#2	*SEG#4*
	LS#2	*SEG#2*
	LS#2	*SEG#5*
	⋮	⋮

Insuffisance des bases de données relationnelles : conclusions

– Il est nécessaire d'avoir la possibilité de décrire des *objets complexes* (*complex objects*) dans une base de données. Un objet complexe est un objet qui n'est pas caractérisé par une seule valeur, mais bien par un ensemble structuré de valeurs.

– Si l'on veut pouvoir faire référence à un objet complexe, par exemple si le domaine d'un attribut de relation est un ensemble d'objets complexes, il faut que les objets aient un *identificateur* (*object identifier*).

Objets complexes et identificateurs d'objet

Il y a plusieurs définitions formelles possibles de la classe des objets complexes. Nous en donnons un exemple typique.

On pose d'abord l'existence des éléments suivants :

- $\{D_i\}$: un ensemble fini de domaines ;
  nous noterons $D = \bigcup_i D_i$;
- A : un ensemble dénombrable d'attributs ;
- ID : un ensemble dénombrable d'identificateurs.

L'ensemble des valeurs d'objets

On définit ensuite l'ensemble des *valeurs* V subdivisé en :

Valeurs de base (*base values*) : ce sont
- *nil* : utilisé pour représenter une valeur non définie ;
- les éléments $d \in D$.

Valeurs-ensembles (*set values*) : ce sont les sous-ensembles de *ID*.

Valeurs-tuples (*tuple values*) : ce sont les fonctions (partielles) de A vers *ID*.

L'ensemble des objets

Un **objet** est une paire (id, v) où

- $id \in ID$ est l'*identificateur* de l'objet,
- $v \in V$ est la *valeur* de l'objet.

Dans tout ensemble d'objets considéré, deux objets distincts ne peuvent avoir le même identificateur.

Les types des objets

On distingue les objets suivant leur type (schéma, structure) :

Objets de base (base objects) : ce sont les paires (id, v) où v est une *valeur de base*.

Le type d'un objet de base (id, v) où $v \in D_i$ est le domaine D_i.
Le type de *nil* est *NIL*.

Objets-ensembles (set objects) : ce sont des paires (id, v) où v est une *valeur-ensemble*.

Le type d'un objet-ensemble (id, v) est 2^T où T est l'union des types des objets (id', v') t.q. $id' \in v$.

Objets-tuples (tuple objects) : ce sont les paires (id, v) où v est une *valeur-tuple*.

Le type d'un objet tuple (id, v) est $\{(a_i, T_i)\}$ tel que $v(a_i)$ est défini et tel que T_i est le type de l'objet (id', v') pour lequel $id' = v(a_i)$.

Les types d'objets : Exemple

Pour définir les routes en tant qu'objets complexes, on considère un ensemble de domaines de base contenant

- D_{lat} : le domaine des latitudes,
- D_{long} : le domaine des longitudes,
- D_{id} : le domaine des identifiants de routes,
- D_{cat} : le domaine des catégories,
- D_{aut} : le domaine des autorités.

Un réseau routier est représenté par des *objets* de la manière suivante.

- Un point d'intersection est un objet tuple de type $T_{point} = \{(\mathit{LATITUDE}, D_{lat}), (\mathit{LONGITUDE}, D_{long})\}$.
- Un segment est un objet tuple de type $T_{seg} = \{(\mathit{POINT1}, T_{point}), (\mathit{POINT2}, T_{point}), (\mathit{CATEGORIE}, D_{cat})\}$.
- Une liste de segments est un objet ensemble de type $2^{T_{seg}}$.
- Une route est un objet tuple de de type $T_{route} = \{(\mathit{ID\_R}, D_{id}), (\mathit{LISTE\_SEG}, 2^{T_{seg}}), (\mathit{AUTORITE}, D_{aut})\}$.

Manipulation des objets et encapsulation

Comment manipuler des objets complexes ? Le modèle orienté-objet adopte l'approche suivante.

- On associe des procédures de manipulation aux objets. Ces procédures sont appelées des *méthodes* (*methods*).

  Exemple : Méthodes qui pourraient être associées à une route.
  - Sélectionner un élément constituant (e.g. les segments, l'autorité).
  - Dessiner la route sur une carte.
  - Modifier la route (e.g. y ajouter un segment).
- Seules les méthodes associées à un objet peuvent être utilisées pour le manipuler **(Principe de l'encapsulation)**.

Les classes

Une *classe* (*class*) est un ensemble d'objets de même type.

- Une classe est l'analogue orienté-objet d'un ensemble d'entités.
- Une classe peut elle-même être un objet (un objet-ensemble). Cela permet de définir des méthodes pour manipuler la classe.

  Exemple : Méthodes qui peuvent être associées à une classe.
  - Modification du contenu de la classe : création ou suppression d'un objet.
  - Traitement de tous les objets de la classe, par exemple sélection de certains objets.

Les hiérarchies de classes et l'héritage

Les objets de certaines classes peuvent être des particularisations d'objets d'autres classes.

Exemple : On pourrait définir une classe *routes* qui se particulariserait en :
- *routes nationales*
- *autoroutes*
  - *autoroutes européennes*
  - *autoroutes nationales*

- Le concept de hiérarchie de classes est anologue au concept de lien "ISA" dans le modèle entités-relations.
- Une sous-classe hérite des attributs (classe d'objets-tuples) et des méthodes de la classe dont elle est une particularisation. Une méthode peut toutefois être redéfinie dans une sous-classe.

Liaison tardive (late binding) et généricité

Dans un langage de programmation classique le lien entre un appel de procédure et le texte du programme correspondant se fait à la *compilation*.

Dans l'approche OO (langages et BD), cela se fait au moment de l'*exécution*.

- Il n'est donc pas nécessaire de connaître le type d'un objet dans un programme qui le manipule.
- Il est donc possible d'écrire des programmes *génériques* pouvant s'appliquer à différentes classes d'objets, ce qui facilite la *réutilisation* de programmes.

Liaison tardive et généricité : exemples

1. Tri générique pouvant s'appliquer à des ensembles d'objets ordonnés de types différents : entiers, chaînes de caractères, paires d'éléments, ...
2. Un programme qui dessine une carte pourrait se réaliser à l'aide d'appels génériques aux méthodes suivantes :
   - dessiner un point d'intersection ;
   - dessiner un segment.

Caractéristiques des bases de données orientées objet

Une base de données orientée objet doit avoir un modèle possédant les caractéristiques suivantes :

- objets complexes,
- identificateurs d'objets,
- héritage,
- encapsulation,
- liaison tardive.

De plus, les possibilités suivantes sont nécessaires à tout système pouvant être considéré comme implémentant une *base de données* orientée objet.

- Complétude au niveau du calcul : toutes les requêtes calculables doivent être expressibles.
- Le système doit être extensible (permettre la définition de nouvelles classes).
- Le traitement de grandes quantités de données doit être possible.
- Une gestion des transactions est nécessaire (parallélisme, récupération après erreur).
- Il doit être possible de poser aisément des requêtes simples.

Langages d'interrogation pour le modèle orienté-objet

Dans ce domaine, il n'y a pas de solution largement répandue. On trouve

- des langages de programmation orienté-objet,
- des langages spécifiques permettant l'interrogation directe de la base de données de façon plus déclarative.

Le modèle objet-relationnel

Combiner les modèles relationnel objet ?

- Le modèle objet permet de traiter des objets complexes et donne de la flexibilité au niveau des types de données possibles.
- Le modèle relationnel est associé à un langage d'interrogation de haut niveau et permet l'exécution efficace de requêtes.
- Combiner les deux modèles permettrait de bénéficier des avantages des deux approches.
- On trouve de plus en plus des extensions "orientées-objet" des modèles relationnels ; SQL-99 prévoit de telles extensions.

Quelles extensions objet ?

Le modèle relationnel-objet est vu comme le modèle relationnel avec des extensions orientées-objet. Les principales sont les suivantes.

- La possibilité, pour l'utilisateur de définir des types. On parle de *User Defined Types (UDT)* ; cela correspond à définir une classe et les méthodes associées. Des hiérarchies sont possibles.
- L'utilisation de références *REF*. Il s'agit d'identificateurs d'objets, mais qui restent visibles.
- La définition de tables dont les lignes sont des objets (plutôt que des tuples). La définition de valeurs "ensemble" pour un attribut est possible.
- La possibilité d'avoir des relations comme valeurs pour certains attributs. On parle de *NESTED TABLES*.

Les requêtes en objet-relationnel

Le langage SQL a été étendu pour permettre la définition et l'interrogation de bases de données objet-relationnel.

- Utilisation de méthodes sur les types définis.
- Suivre les références (*DEREF*).
- Interrogation des tables imbriquées : *THE()* permet de traiter une table imbriquée comme une table ordinaire.