Des Relations aux Objets : Introduction aux SGBD objets

Bernard ESPINASSE
Professeur à Aix-Marseille Université (AMU)
Ecole Polytechnique Universitaire de Marseille
Avril 2013

- **Forces et faiblesses du modèle Relationnel**
- **L'approche orientée objet et son intérêt pour les bases de données**
- **Les options d'évolution des BDR vers l'objet**
- **L'approche Wrapping Objet**
- **L'approche Révolutionnaire (SGBD Orientés Objets) : l'exemple de O2**
- **L'approche Evolutionnaire (SGBD Objet-Relational) : l'exemple de POSTGRES**
- **Conclusion**

Plan

Bibliographie du cours

Livres :

- Gardarin G., « Bases de données objet et relationnel », Ed. Eyrolles, 1999 (ISBN : 2-212-09060-9).
- …

Cours de :

- Georges Gardarin
- Didier Donsez
- Christophides Vassilis
- …

Introduction

- **Forces et faiblesses du modèle Relationnel**
- **L'approche orientée objet et son intérêt pour les bases de données**
- **Manifeste pour une 3ième génération de SGBD**
- **Les options d'évolution des BDR vers l'objet**

SGBD Relationnels : quelques dates...

- *1966: commercialisation IDMS & IMS*
- **1970: invention du modèle relationnel (CODD)**
- *1971: recommendations CODASYL*
- **1981: début des projets Système R et Ingres**
- **1986: normalisation de SQL1**
- **1990: extentions aux objets et règles**
- **1990: normes SQL2**
- **1999: normes SQL3 (SQL 99)**
- **200X : SQL4 ?**

Forces du modèle relationnel

- **modèle simple**
- **approche formellement définie (normalisation, algèbre, ...)**
- **Langage de Manipulation de Données (LMD) déclaratif (Standard SQL 2)**
- **niveau logique (essentiellement)**
- **technologie la plus répandue**
- **efficace pour les applications de gestion classique**

Faiblesses du modèle relationnel

- **modèle sémantiquement pauvre :**
  - **difficile de modéliser des structures complexes** : pas d'attributs complexes ni multi-valués, un seul type de lien (clé étrangère) : jointure indispensable
  - **un seul type de lien (clé externe)**
  - **le système de types figé (non extensible)**
- **Langage de Manipulation de Données non complet**
  - **(au sens de Turing): recours à un langage de programmation**
- **Séparation données-traitements : description statique et non dynamique du monde**
- **Dysfonctionnement entre SQL et les langages de programmation traditionnels ("impedence mismatch") :**
  - **2 philosophies différentes (déclaratif/procédural et ensembliste/un élément à la fois)**
  - **2 mondes obligés à communiquer (conversions à effectuer)**

Faiblesses du modèle relationnel (suite)

- **Données alphanumériques uniquement (pas d'images, de sons, de vidéo, ...)**
- **Contrôle de concurrence exclusivement orienté gestion**
- **Absence de mécanisme de versions**

Ainsi mal adapté à :

- **de nouvelles applications : aide à la décision, conception, géographie, bureautique, SIG, multimédia, ...**
- **de nouvelles techniques : IHM, programmation orientée objet, programmation en logique, architectures réparties, ...**

L'approche orientée objet

- **{méthodologies et d'outils} pour concevoir et réaliser des logiciels structurés et réutilisables, par composition d'éléments indépendants**
  - productivité des programmeurs
  - réutilisation
- **Concepts essentiels**
  - **objet encapsulé**
    - interface visible : opérations (méthodes)
    - implémentation cachée : structure et code
  - **héritage**
- **Langages de programmation OO**
  - Eiffel, Smalltalk, C++, Java …

Intérêts de l'approche objet pour les BD

Identité d'objet :
- Favorise le **partage de données**
- Supporte des **pointeurs typés**

Encapsulation de données :
- Permet **l'isolation de données des opérations**
- Facilite **l'évolution des structures de données**

Héritage d'opération et de structure :
- Facilite la **réutilisation de type de données**
- Permet de **particulariser les programmes aux besoins de l'application**

Possibilité de définir des opérations abstraites (polymorphisme) :
- **Augmente la productivité** des développeurs d'applications

Manifeste pour une 3ième génération de SGBD 1990

Supporter des structures d'objet complexes et des règles :
- Système de types riche, héritage, encapsulation
- Fonctions,optionels/unique ids, règles/triggers

Subsumer le 2ième génération de SGBD :
- Interface d'interrogation de haut niveau
- Collections de données stockées ou virtuelles
- Vues modifiables
- *Data model/performance feature separation*

Ouverture vers d'autres systèmes (outils, middlewares, …) :
- Etre accessible de différents langages
- *Layered persistence-oriented language bindings ;*
- *Query-shipping architecture*

Les options d'évolution des BDR vers l'objet

Encapsuler le SGBD Relationnels
- Wrapper client Orienté Objet (**Wrapping Objet**)
- Langage de Programmation Orienté Objet + SGBDR

Construire une nouvelle technologie de BD
- SGBD Orienté Objet (**SGBD-OO**)
- Langage de Programmation Orienté Objet persistant

-> Approche révolutionaire

Construire une nouvelle technologie de BD
- SGBD Objet-Relationnel (**SGBD-OR**)
- Modèle Relationnel + extension orientées objet

-> Approche évolutionaire

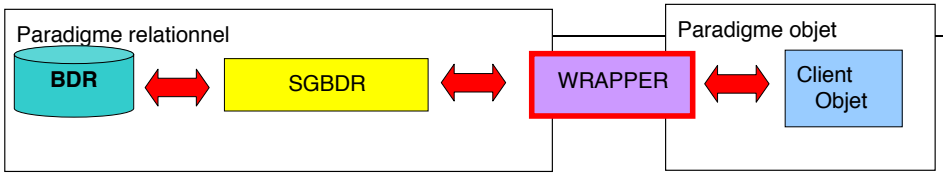
Wrapping Objet

Offre commerciale :

- Ardent, Persistence Software, Ontologic, …
-

Wrappers relationnels à base de langages specific :

- Proxi classes (C++/Java)
- Mapping de tuple dans le langage objets
- Exécution côté client de méthodes d'extraction d'objet



Wrapping Objet : forces et faiblesses

Forces :

- Permet utilisation de systèmes existants
- Développement rapide d'application orientées objet exploitant les données (business objects)

Faiblesses :

- Cohabitation de paradigmes différents :
  - serveur : SQL(Schéma Relationnel)
  - client : C++ ou Java (Schéma objet)
- Pas d'intégration au SGBDR, ne l'utilise pas pour le maintien de l'intégrité, procédures stockées, … dommage !!!

⇨ **Solution limitée**

BdD Objets : Approche Révolutionnaire

Approche Révolutionnaire (SGBD Orientés Objets)

- Bancilhon & al. 89 : **"Object-Oriented DBMS Manifesto",** DODD Conf., dec 89
  - objet = modélisation d'une entité observable du réel constituée de :
    - d'attributs : propriétés construites à partir de types primitifs
    - de relation(s) : propriétés dont type = référence à un ou une collection d'objet(s)
    - de méthode(s) : fonctions pouvant être appliquées à l'objet
- **Produits** : **O2**, **Gemstone**, **Orion, Encore, …**
- **Langage standard** : **OQL (ODMG)**

BdD Objets : Approche Evolutionnaire

Approche Evolutionnaire (SGBD Objet-Relational)

- ACM 89 : **"Third-generation DBMS manifesto", 3 principes d'intégration:**
  - intégration des nouveaux concepts **d'objets** et **règles** (encapsulation, héritage, fonctions, déclencheurs, ...)
  - maintient de l'approche ensembliste, SQL, mises à jour sur vues, …
  - intégration dans les architectures ouvertes et standardisées (client-serveur)
- **Produits** : **Postgres**, IBM **DB2 CS/UDB**, **CA-Ingres, Illustra, UniSQL/X, Oracle 8**, **Informix, …**
- **Langage standard** : **SQL3 (ANSI X3 H2)**

SGBD Orientés Objets

- Les 8 règles d'or d'un SGBD-OO
- Le SGBD Orienté Objet « O2 »

Les 8 règles d'or d'un SGBD-OO

Fondements :

- Bancilhon & al.89 : "**Object-Oriented DBMS Manifesto**", DODD Conf., dec 89

Services langage	Services BD
R1 : Identité d'objet (ID) mise à jour, partage, graphes sémantique d'ID	**R5 : Persistance** modèle formel, algèbre, gestion mémoire virtuelle (garb.collec.)
R2 : Abstraction (types/classes), liaisons dynamiques	**R6 : Structuration complexe** (attributs et entités), constructeurs (tuples, set), schéma
R3 : Encapsulation (méthodes, messages)	**R7 : Sécurité** confidentialité, intégrité (déclencheurs, etc...), synchronisation ; reprise après panne, transactionnel
R4 : Réutilisation héritage (simple, multiple), polymorphisme (surcharge), versions	**R8 : Interfaces non procédurales** traitements ad hoc, optimiseur

Identité (ID)

- **un objet existe s'il est identifiable (OID)**
- cette identification **est un invariant dans la vie de l'objet**
- **l'existence** d'un objet est **indépendante de sa valeur**

En conséquence il est possible de :

- **référer un objet** de manière **non ambiguë** par son identifiant
- **mettre à jour la valeur** d'un objet **sans remettre en cause son existence**
- **représenter des liens sémantiques entre objets** (liens entre identifiants)
- **distinguer l'identité de l'égalité** (mêmes valeurs mais identités différentes)
- **distinguer des copies "profondes"** (concernant tous les sous-objets) ou "**superficielles**" (concernant seulement les références)

Abstraction

- possibilité d'éliminer les caractéristiques non essentielles d'un objet de façon à créer des regroupements d'objets appelés "**types**" ou "**classes**"
- un **objet** est alors une **instance** d'un **type** ou d'une **classe**

Encapsulation

- le **type** est défini avec tous les **opérateurs** spécifiques qui lui sont **applicables**
- ces opérateurs sont souvent appelés "**méthodes**"
- l'appel d'une "méthode" se fait par envoi d'un "**message**"
- seul de l'ensemble des opérateurs du type, la "**signature**" est **visible** pour **l'utilisateur**, l'implantation des opérateurs est cachée.

Réutilisabilité

- **Polymorphisme (l'encapsulation):** une fonction polymorphe (ex: afficher) peut s'appliquer à une divers d'objets (image, texte...)
- **Héritage (abstraction - sous typage) :**
  - permet de factoriser entre objets soit des structures de données, soit des opérateurs
  - peut être simple, multiple, sélectif, partiel

Persistance

- Quels objets persistent dans la BD à la suite de l'application qui les a fait naître ?

Structuration complexe

- BDR : impossible de créer de nouveaux types à partir des types prédéfinis
- BDO : des constructeurs permettent l'extensibilité des types de base

Le SGBD Orienté Objet « O2 »

Fonctionnalités :

- un SGBD Orienté Objet : **O2 Technology** (à l'origine startup INRIA)
- un environnement de programmation complet : débogueur, navigateur/éditeur dans le schéma et dans la base, une boite à outils : **CO2, O2Query, OQL, …**
- des outils de génération d'interfaces : **O2LooK**

Modèle de données :

- gère les valeurs complexes
- distinction entre objets-valeurs
- les classes et les méthodes ne sont pas des objets
- l'ensemble est un constructeur de classe
- les objets sont fortement typés
- sémantique de l'inclusion pour l'héritage
- les objets nommés sont les points d'entrée de la BD et les racines de la persistance

O2: Types et Valeurs

Valeurs :

- *tuple* (nom: "Dupont",
  age: 41,
  marie: true,
  enfants: [*tuple* (prenom: "Pierre", age: 18),
  tuple (prenom: "Jeanne", age: 14)],
  voiture: { *tuple* (marque: "renault", modele: "R5", couleur: "rouge"),
  tuple (marque: "volvo", modele: "240", couleur: "blanche") })
- { [1, 2, 3] , [3, 2, 1] , [2, 3, 1] , [3, 1, 2] }

- les valeurs atomiques : *25, 41 , "Pierre", true* - la valeur nulle : *nil*
- les constructeurs :
  - **ensemble** : set (1,2, 3) ; **n-uplet** : tuple (nom: "Pierre",age: 18) ; **liste** : list (12, 14, 12)
  - orthogonaux : on peut avoir des listes d'ensembles de listes de n-uplets, ...

Types :

- le **type** spécifie le type de la valeur des objets :
  - *tuple* (nom: *string*,
    age: *integer*,
    marie: *boolean*,
    enfants: *list (tuple* (prenom: *string* , age: *string*),
    voiture: *set (tuple* (marque: *string* , modele: *string* , couleur: *string*),
  - *set (list (integer))*

Un objet dans O2

un objet = un identificateur + une valeur + {méthodes}

- les **valeurs** peuvent utiliser les **objets** comme des **valeurs atomiques**
- une **méthode** :
  - prend en entrée des **objets** et des **valeurs** et rend un **objet** ou une **valeur**
  - est définie par son **corps** et sa **signature** (= **liste** des **arguments d'entrée** et de **sortie** avec leur **type**)

(**tourEiffel**,
tuple (nom: "la tour Eiffel",
adresse: tuple (numéro: 1, rue: "champ de Mars", **ville** : Paris),
prix: 28),
anneeConstruction: 1889),
age: *integer*,
augmenterPrix (*integer*) : Monument)

l'objet Paris peut être partagé par d'autres objets

Données

Méthodes

- il existe un objet **Nil** ayant la valeur "nil"

O2: une classe

• les **classes** peuvent être utilisées comme des **types atomiques**

```
classe   Hôtel
type     tuple (nom:   string,
                adresse:  tuple (numéro:  integer,  rue:  string,  ville: Ville),
                étoiles:  integer)
method          nombreDEtoiles:   integer;

classe   Ville
type     tuple (nom:  string,
                pays:  Pays,
                carte:  Carte,
                monuments:   set  (Monuments),
                hôtels:  set  (Hôtel),
method          nombreDeMonuments:   integer;
```

• il existe une classe spéciale : la **classe "Object"** qui est de type "**Any**"

O2: Objet & Valeurs

Les Objets :

- ont une **identité**
- appartiennent à une **classe** (équivalence par nom)
- sont manipulés par des **méthodes**

Les Valeurs :

- n'ont **pas d'identité**
- sont d'un **type** (équivalence par structure)
- sont **manipulées par des primitives**

O2: héritage

• soit 2 classes : C1= (n1, t1, M1) et C2= (n2, t2, M2), si l'utilisateur définit C2 comme une sous-classe de C1 :

- t2 est un **sous-type** de t1
- M2 est **compatible** avec M1 et dans M2, des méthodes peuvent être ajoutées ou redéfinies

```
classe   AVisiter
type     tuple          (nom:   string,
                adresse:  tuple (numéro:  integer,  rue:  string,  ville: Ville),
                étoiles:  integer)
    method    description: string;
          ville : Ville;

classe   Monument inherits   AVisiter
type     tuple          (prix:   integer,
                annéeConstruction:   integer),
method âge: integer;
          augmenterPrix  (integer) :   Monument
          description:   string;
```

• pour l'**héritage multiple**, les conflits sont **résolus par l'utilisateur**

O2: Persistance

• **objets** et **valeurs** peuvent être **nommés** :

```
name     tourEiffel:     Monument;
name     monumentsParisiens:     set (Monument);
```

• les noms permettent de définir la persistance :

- **tout objet(valeur) nommé est persistant**
- **tout objet(valeur), élément d'un objet(valeur) persistant est persistant**

• les objets sont créés par une primitive :

- **new nom-classe** : tourEiffel = *new* (Monument);

```
o2     Ville     v;
v  =  new  Ville;
tourEiffel -> adresse.ville  =  v;
      "tourEiffel" est un objet persistant car nommé
      "v" devient persistant car accessible à partir d'un objet persistant
```

• les objets ne peuvent être détruits explicitement, seuls les liens sont supprimés

• lorsqu'un objet n'est plus rattaché à un objet persistant (directement ou non), il disparaît :

```
tourEiffel -> adresse.ville  =  v';
      si v n'est plus référencé par aucun objet, il disparaît
```

O2: Le langage O2C

extension naturelle de C

- permet de **créer des objets O2**, de leur **envoyer des messages** et de **manipuler des valeurs complexes**
- constitue un **L4G** (prog., manipulation de BD, IHM)

```
o2   Monument   arche;                    déclarations
o2   set (Ville)   villesItaliennes;
villesItaliennes = set ( );                manipulation de valeurs
villesItaliennes += set (Venise);
for   (x in villesItaliennes)
      printf ("%s a %d monuments \n", x->name,
       x->nombreDeMonuments() );

public method   pasCher:   boolean in class Monument;
method body     pasCher: boolean in class Monument
      {return (*self .prix <20);
      }
```


O2: Le langage O2Query

langage de requêtes (=SQL objet):

tourEiffel.adresse — *adresse de la tour Eiffel*

select tuple (nom: h.nom, adresse: h.adresse) ***from*** h ***in*** Hôtels

where h.ville.nombreDeMonuments > 10 ***and*** h.étoiles = 2;

nom et adresse des hôtels 2 étoiles des villes ayant au moins 10 monuments

SGBD Objet-Relationnel

- Modèle Objet-Relationnel
- Le SGBD Objet-relationnel « POSTGRES»

Approche Evolutionnaire (SGBD Objet-Relationnel)

Fondements :

- **ACM 89 : "Third-generation DBMS manifesto"**

3 principes d'intégration:

- intégration des nouveaux concepts d'**objets** et **règles** (encapsulation, héritage, fonctions, déclencheurs, ...)
- maintient de l'approche **ensembliste**, **SQL**, persistance des données, mises à jour sur vues
- **intégration** dans les architectures ouvertes et standardisées (client-serveur)

Produits :

- **POSTGRES**, **IBM DB2 CS (V2.1), IBM DB2 UDB, CA-Ingres, Illustra, UniSQL/X, Oracle 8**, **IBM DB2 UDB, Informix, …**

Modèle Objet-Relationnel

La relation est l'abstraction fondamentale (pas la classe comme pour les SBBD OO)

Extension du modèle relationnel :

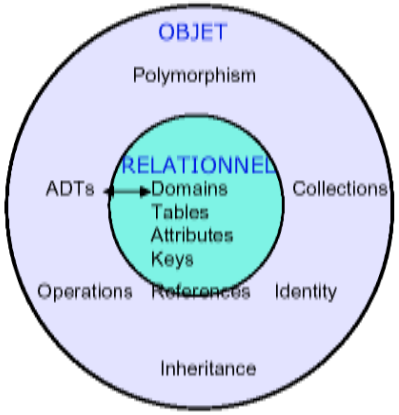
- attributs structuré et multi-valués
- héritage pour les relations et les types
- ADTs (Types abstraits de données) pour les domaines
- Identification d'objet pour les tuples
- Surcharge d'opérations

Extension de SQL :

- Schémas : tables au sommet et ajouts OO à l'intérieur
- Requêtes : extension pour supporter les ajouts OO

Modèle Objet-Relationnel

(Vassilis)



Modèle Objet-Relationnel : Exemple

Tables & Objets: Exemple (Oracle8)

(Vassilis)

Name	Style	Live-Time	Influences		Artifacts			
			Name	Date	Title	Material	Photo	Descript
Claude Monet	Impres-sionism	1840-1926	Edouard Manet	1863	Haystacks at Chailly at Sunrise	Oil on Canvas		
			Eugene Boudin	1864	Wheat-stacks End of Summer	Oil on Canvas		
					Meule, Soleil Couchant	Oil on Canvas		

SGBD Objet-Relationnels disponibles

Systèmes disponibles :

- **IBM DB2 CS (V2.1) and CA-Ingres**
  - Types et fonctions définis par l'utilisateur, larges objets, triggers
- **Illustra, UniSQL/X**
  - Fournisseurs de ADTs, objets tuples, héritage,
- **IBM DB2 UDB, Informix, Oracle 8**
  - Fournissent des ous-ensemble de toutes ces fonctionnalités
- **POSTGRES**
  - Developpé à l'Université de Berkeley (Stonebraker & al.)

POSTGRES : introduction

- Projet universitaire dirigé par Stonebraker, Rowe 1986-1994. (Post-INGRES Université de Berkeley)
- commercialisé sous le nom de **ILLUSTRA** racheté par INFORMIX IUS

Le modèle de données de Postgres :

- **Terminologie objet :**
  - class = relation ; instance = tuple ; object-id = tuple-id
  - method = attribut or function d'attributs
- **Support des ADTs extensibles :**
  - procédures extensibles utilisant des functions C
  - opérateurs binaires
- **Support des constructeurs de types :**
  - utilise des requêtes
  - supporte directement les tableaux (Array)
- **Héritage de classes** : permet un héritage de méthodes et une collection de hiérarchies

POSTGRES : classes & types

CLASSE (analogue à une relation) **: collection d'instance d'objets ayant des attributs d'un certain type**

- chaque objet a un identifiant unique = **OID**
- **héritage multiple** entre les classes (pas de résolution de conflits)

3 sortes de CLASSES :

- **real** ou classes de base (stockées dans BD)
- **derived** ou classes virtuelles (views) définies par des règles
- **version** d'une autre classe (differential)

Système extensible de TYPES avec 3 sortes de type :

- **base types** : prédédinis ou définis par l'usager (ADT)
- **array of base types** (tableaux, listes)
- **composite types** ou types complexes ("class et set") :
  class : zéro, une ou plusieurs instances d'une classe
  set : collection d'objets pouvant appartenir à différentes classes

Postgres: classes & types

Déclarations de classes :

- create EMPLOYE (nom=c12, salaire=float, age=int)
- create VENDEUR (quota=float) inherits EMPLOYE
- create DEPT (nomd=c10, directeur=c12, plan= polygone, bp=point) avec polygone et point = types définis

Affectation de constantes ou de valeurs retournées par des fonctions :

- replace DEPT (bp="(10,10)") where DEPT.nomd="jouets") ;
- replace DEPT (bp=center(DEPT.plan)) where DEPT.nomd="jouets") ;

Tableaux de types de bases :

- create EMPLOYE (nom=c12, salaire=float[12], age=int)
- retrieve (EMPLOYE.nom) where EMPLOYE.salaire[4]=10000)
- retrieve (EMPLOYE.nom) (salaire[6]=salaire[5]) where EMPLOYE.nom="Dupont"

Postgres: classes & types

Objets complexes :

class :

- create EMP (nom=c12, salaire=float, age=int, directeur=EMP, collegues=EMP)

set :

- add to EMP (hobbies=set)

Expressions de chemins et notation pointée "." :

- retrieve (EMP.directeur.age) where EMP.nom="Dupont"
- replace EMP (hobbies=calcul_hobbies ("Dupont")) where EMP.nom="Dupont" ; calcul_hobbies est une fonction spécifique de Postgres

Postgres: 3 types de fonctions

C functions :

• les arguments = types de base ou types composés (ex: nom de classe) :

retrieve (DEPT.nomd) where **area** (DEPT.plan)>200 ;

fonction **area** : polygone -> float

retrieve (EMPLOYE.nom) where **hautsal**(EMPLOYE) ;

fonction **hautsal** : EMP -> boolean (par exemple vrai si le salaire est > 30000F)

• sont des **méthodes** et peuvent ainsi être **héritées**

• peuvent aussi être considérées comme des attributs :

retrieve (EMPLOYE.nom) where EMP.hautsal

Operateurs :

• fonctions **avec 1 ou 2 opérandes** qui utilisent la notation des opérateurs du langage

• ne peuvent s'appliquer que sur des **types de base**

retrieve (DEPT.nomd) where DEPT.plan **AGT**"(0,0), (1,1), (0,2)": AGT= "area greater than"

• les opérateurs sont codés par le "database implementator"

POSTQUEL functions : tout ensemble de requête (d'interrogation) en POSTQUEL

Postgres: fonctions POSTQUEL

POSTQUEL: langage relationnel ensembliste augmenté de possibilités pour :

• les requêtes imbriquées, la fermeture transitive, le support de l'héritage, l'interrogation temporelle

POSTQUEL functions :

• tout ensemble de requête (d'interrogation) en POSTQUEL :

define function **cadre** returns EMP as retrieve (EMP.all) where EMP.salaire>40000

define function **rech_sal (c12)** returns float as retrieve (EMP.salaire) where EMP.nom= **$1** ; possibilité de paramètres (ici le nom de l'employé)

• peuvent avoir une classe comme argument, fonction alors définie sur chaque instance de la classe :

define function **voisins (DEPT)** returns DEPT as retrieve (DEPT.all) where DEPT.étage=*.étage

• peuvent être utilisées dans les requêtes :

retrieve (DEPT.nomd) where **voisins** (DEPT).nomd="jouets"

• peuvent être vues comme de nouveaux attributs :

retrieve (DEPT.nomd) where DEPT.**voisins**.nomd="jouets" ; on cherche les dept qui sont au même étage que jouets

Postgres: langage de requête

Imbrication :

retrieve (DEPT.nomd) where DEPT.etage NOT-IN {D.etage from D in DEPT where D.nomd≠DEPT.nomd}

Fermeture transitive :

class parent(vieux, jeune)

retrieve * into reponse (parent.vieux) from r in reponse where parent.jeune="bernard" or parent.jeune=r.vieux

la requête s'exécute tant qu'elle produit de nouveaux résultats,(*), le résultat peut être rajouté à la base comme ici, les doubles sont éliminés automatiquement

Héritage :

* après le nom d'une classe = toute la hiérarchie d'héritage

retrieve (e.nom) from e in EMP* where e.age>40 ; {tous les employés de plus de 40 ans}

Requêtes temporelles :

retrieve (EMP.salaire) from EMP[t] where EMP.nom="bernard" ; salaire de bernard à l'instant t

Postgres: langage de règles

Règles de production permettant de traiter : la gestion des vues, les "triggers" ou déclencheurs, les contraintes d'intégrités, l'intégrité référentielle, la protection, le contrôle de processus

ON événement (TO) objet
WHERE qualification_postquel
THEN DO [instead] commande(s)_postquel

- événement : retrieve, replace, delete, append, new (pour replace ou append) et old (pour delete ou replace)
- objet : nom de classe ou nom de classe.nom_d'attribut
- qualification_postquel : idem langage de requête
- instead : facultatif, indique que les commande(s)_postquel doivent être exécutées à la place de l'action qui a déclenchée la règle. Si pas mentionné, exécutées en plus de l'action qui a déclenchée la règle
- commande(s)_postquel : {commandes postquel} avec 2 changements suivants :
  - "new" ou "current" peuvent apparaître à la place d'un nom de classe devant tout attribut
  - refuse (target_list) est ajoutée comme nouvelle commande.

Postgres: langage de règles

Exemples :

- on new EMP.salaire where EMP.nom="bernard" then do replace E.salaire=new.salaire from E in EMP where E.nom = "sabine" ;

les modifications de salaire de bernard sont à propager sur celui de sabine

- on retrieve to EMP.salaire where EMP.nom="sabine" then do instead retrieve (EMP.salaire) where EMP.nom="bernard" ;

le salaire de sabine est le même que celui de bernard (les 2 salaires non pas besoin d'être stockés, celui de sabine sera dérivé à partir de celui de bernard)

Note :

si le salaire de bernard est lui même calculé, il faudra déclencher d'autres règles ("backward chainning"). Il existe aussi la stratégie "forward chainning".

Postgres: les vues

Extension de la notion de vue relationnelle définies par règles

La définition d'une vue V est compilée en un ensemble de règles pour gérer V (accès et mises à jour)

define view EMP_JOUETS (EMP.all) where EMP.dept="jouets"

compilée en :

on retrieve to EMP_JOUETS (EMP.all) then do instead retrieve (EMP.all) where EMP.dept="jouets"

Default view

define default view SMIG (EMP.OID, EMP.nom, EMP.age where EMP.salaire < 5000

règle générée pour replace :

on replace to SMIG.age then do instead replace EMP(age=new.age) when EMP.OID=current.OID ;

on modifie le n-uplet stocké qui est repéré par son OID

Postgres: les vues

Vue comme une jointure et mise à jour :

define vue ED (EOID=EMP.OID, EMP.name, EMP.salaire, EMP.dept, DOID=DEPT.OID, DEPT.etage) where EMP.dept=DEPT.nomd

on retrieve to ED then do instead retrieve (EOID=EMP.OID, EMP.name, EMP.salaire, EMP.dept, DOID.OID, DEPT.etage) where EMP.dept=DEPT.nomd

Mise à jour du nom et du salaire de l'employé :

on replace to ED.name, ED.salaire then do instead

replace EMP (nom=new.nom, salaire=new.salaire) where EMP.OID=current EOID

Mise à jour du département d'un employé, si le département donné n'existe pas, on le crée :

on replace to ED.dept then do instead

replace EMP (dept=new.dept) where EMP.OID=current.EOID

append to DEPT(nomd=new.dept, etage=current.etage) where new.dept not in {DEPT.nomd}

Conclusion sur les SGBD orientées objets

Les SGBD Relationnels traditionels (SQL2) sont mal adaptés :

- aux nouvelles applications : aide à la décision, conception, géographie, bureautique, SIG, multimédia, …
- aux nouvelles techniques : IHM, programmation orientée objet, programmation en logique, architectures réparties, ...

Les SGBD Orientés Objet (OO) :

- remettent trop en cause l'existant
- ne tirent pas suffisamment profit des investissements considérables réalisés dans le relationnel

Les SGBD Objet-Relationnel (OR) :

- s'appuient bien sur ces investissements
- disposent d'un langage maintenant normalisé **SQL3**
- nombreux grands opérateurs l'ont déjà adopté (IBM, Oracle, …)
- **ils devraient assez rapidement s'imposer dans les entreprises**