Norme SQL3 et Objet-relationnel

Anne-Cécile Caron

Master MIAGE - BDA

1er trimestre 2010-2011

Pourquoi le modèle objet

- Identité d'objet : Modèle orienté objet par opposition à un modèle orienté valeur.
- objets complexes : attributs multi-valués, collections.
- Encapsulation des données.
  - Accès aux données via des opérations
  - Structure de données cachée et donc plus facilement modifiable.
- Héritage d'opérations et de structures

SGBDOO

En base de données, l'instance aura des caractéristiques différentes du modèle relationnel :

- Dans le schéma, les associations sont orientées (rôles en UML) → l'instance de la base est un graphe orienté.
- Les requêtes permettent de **naviguer** dans cette instance, à partir de points d'entrées
- Les instances d'une même classe sont (souvent) stockées dans une même collection.

Applications :

- Données dont la structure est complexe,
- ressemblances entre les données (hiérarchie de classes facile à construire)
- relation composé/composant ...
- niches technologiques :
  - Gestion de données techniques
  - Systèmes d'informations géographiques
  - CAO
  - ...

Pourquoi objet-relationnel ?

- SGBD objet :
  - modèle objet *pur* donc plus élégant.
  - pas vraiment de SGBDOO qui ont les performances des SGBDR.
  - norme ODMG en 1993 (le groupe ODMG s'est dissout en 2001), sans succès.
- SGBD objet-relationnel :
  - concilier un modèle orienté objet avec un modèle orienté valeur.
  - continuité, SGBD relationnels déja bien implantés.
  - norme SQL3

La norme SQL 3

aussi appelée SQL'99

- Extension du modèle relationnel à l'objet
  - nécessité de structures de données plus sophistiquées pour certaines applications.
  - Echec des SGBD objets (norme ODMG)
- On peut toujours faire du relationnel : enrichissement de SQL2 (1992)
- Beaucoup plus complexe que les normes précédentes environ 2100 pages contre 150 pour SQL'89
- en 2003 et 2006 : ajout de SQL/XML, manipulation de données XML
- en 2003 : SQLJ (SQL intégré à java)

Que contient cette norme ?

- Un langage de définition de types (User Defined Types)
- Un langage de requêtes
  - SQL est étendu au modèle objet-relationnel.
  - SQL est étendu aux structures récursives (i.e. une pièce composée d'autres pièces),
  - on peut aussi utiliser les expressions régulières pour les sélections.
- Un langage de programmation (procédures stockées, triggers,méthodes)

Les Types

- Types prédéfinis simples
  - ceux de SQL2
  - Types intervalles de temps
  - type BOOLEAN (true, false, unknown)
- Types collections (ARRAY)
- Types lignes (ROW)
- Large objects : BLOB, CLOB

Types ARRAY

- Tableau à une dimension, de taille variable (mais bornée)
- Tous les éléments sont du même type.
- Constructeur `ARRAY[]`
- pas autant de possibilité qu'en objet (set, multiset, list)

```
CREATE TABLE Mailouts{
      given_name VARCHAR(15),
      surname VARCHAR(20),
      street_address VARCHAR(20) ARRAY[5]);

      INSERT INTO TABLE Mailouts(given_name, surname, street_address)
      VALUES ('Jean','Dupont',ARRAY['line1','line2','line3']);

      UPDATE Mailouts SET street_address[2]='blabla'
      WHERE surname = 'Dupont';
```

Types enregistrements : ROW

les types ROW permettent de représenter les lignes des tables.

- Pour toute table, une ligne est d'un type ROW (induit par le schéma de la table)
- défini par une liste de couples (nom de champ,type).
- Le type d'un nom de champ peut lui-même être de type ROW.
- Constructeur ROW(liste de valeurs)
- Accès aux champs par notation pointée.
- comparaison prédéfinie (ordre lexicographique sur les champs)

Types ROW (2)

```
CREATE TABLE employees (
      name VARCHAR(40),

      address ROW(street VARCHAR(30),
                  city VARCHAR(20),
                  zip VARCHAR(10) )
);

INSERT INTO employees
VALUES('John Doe', ROW('2225 Coral Drive', 'San Jose','5678967'));

UPDATE employees SET address.street = 'blabla'
WHERE name = 'John Doe' ;
```


User Defined Types (UDT)

Types distincts ou types structurés

- type distinct = basé sur un type prédéfini

```
CREATE TYPE <UDT name>
AS <predefined data type> FINAL ;

create type chaine15_udt as CHAR(15) FINAL ;
create type euro_udt as DECIMAL(9,2) FINAL ;
create type franc_udt as DECIMAL(9,2) FINAL ;
```

- type structuré = type objet (ADT)

Les Types structurés ou ADT

- La définition d'un ADT comporte des **attributs** et des **méthodes**.
- La définition d'un attribut entraîne la création d'une méthode de mise-à-jour (mutator method) et d'une méthode d'accès (observer method)
- un ADT peut typer un attribut d'ADT, une colonne d'une table, une ligne d'une table ...
- On peut définir une relation d'ordre sur un ADT.
- On peut définir des fonctions de conversion de types.

Exemple d'ADT

```
Create type book_udt AS
  title VARCHAR(30),
  buying_price DECIMAL(9,2),
  selling_price DECIMAL(9,2)
  NOT FINAL                 /* peut être sous-typé */
  METHOD profit() returns DECIMAL(9,2) ;

Begin
  declare u book_udt ;
  declare c varchar(30) ;
  set u = book_udt() ;  // constructeur
  set u = u.title('le titre du livre'); // mutator method
  ...
  set c = u.title() ; // observer method
  ...
end ;
```


Exemple d'ADT (suite)

- Typer la colonne d'une table :

```
Create table T(
  book_column book_udt,
  serial_number INTEGER
);

insert into T values (u,1) ;
où u est un objet de type book_udt.
```

- Typer les lignes d'une table (*table objet*) :

```
Create table BOOK of book_udt ;

insert into BOOK(title,buying_price,selling_price)
values ('SQL3 pour les nuls',10.0,20.0) ;

select selling_price - buying_price
from book
where title like '%SQL%' ;
```


Héritage

- Héritage simple d'ADT : Le sous-type hérite tous les attributs et méthodes de son super-type. Le sous-type peut aussi définir d'autres attributs et méthodes, et redéfinir des méthodes.
- Hiérarchie de tables : La relation table/sous-table doit suivre la relation type/sous-type.

```
Create type  science_book_udt under book_udt AS
  domain VARCHAR2(30),
  key_words CHAR(10) ARRAY[5] ;

Create table SCIENCE_BOOK
of science_book_udt
UNDER book ;
```


Types Références

Référence (pointeur) sur une ligne de type structuré (ADT).

```
REF <udt name> [SCOPE <table name> [<reference scope check>]]

<reference scope check> ::= REFERENCES ARE [NOT] CHECKED
                            [ON DELETE {CASCADE | SET NULL |
                                SET DEFAULT | RESTRICT | NO ACTION}]

Create table emprunte(
  the_book REF(book_udt) SCOPE book
           REFERENCES ARE CHECKED ON DELETE CASCADE,
  the_person REF(person_udt) SCOPE person ,
  beginning DATE,
  end DATE
  ...
);
```

Comparaison avec les Modèles Objets

- Classes : Les ADT sont des classes.
- Encapsulation :
  - pas de notion de visibilité (private, public ...)
    c'était pourtant prévu au départ.
  - Méthodes de mise-à-jour et méthodes d'accès.
  - Gérer les droits via les commandes SQL.
- Types collections : uniquement ARRAY
  au départ : SET, LIST, MULTISET.
- Héritage : hiérarchie d'ADT, hiérarchie de tables.
  Au départ : héritage multiple d'ADT et de tables.
- Une table typée par un ADT = extension d'une classe.
- Introspection : utiliser le dictionnaire.

Extensions du langage de requête

- Union Récursive : Pour parcourir une association réflexive (ex : une pièce formée d'autres pièces)
- Group by : nouvelles façons de fabriquer les groupes
- Navigation : accès direct à l'information via les REF, sans utiliser de jointures.

```
Select e.the_book.title from emprunte e
where e.the_person.name = 'SMITH' ;
```

- Appel de fonction, appel de méthode.
- Comment explorer les collections imbriquées (ARRAY) ?
- expressions régulières

Les triggers

- Pas présents dans la norme SQL2
- Mais déjà présents dans de nombreux SGBD.
- Très proches des Triggers d'Oracle :
  - lié à une table
  - `BEFORE/AFTER`
  - trigger ligne ou trigger instruction (par défaut)
  - clause `WHEN`
  - MAIS limitations dans le corps du trigger (par exemple : pas de variables hôtes).

Les triggers (suite)

```
CREATE TRIGGER <trigger name>
{BEFORE|AFTER} <trigger event> ON <Table name>
[REFERENCING <old or new values alias list> ]
[FOR EACH {ROW | STATEMENT}][WHEN (search condition)]
<triggered SQL statement>

<trigger event> ::= INSERT | DELETE | UPDATE [OF <column list>]

<old or new values alias list> : permettent de donner un alias
à l'ancienne et la nouvelle ligne (trigger for each row)
à l'ancienne et la nouvelle table (trigger statement)

<triggered SQL statement> ::= SQL statement
                            | BEGIN ATOMIC {SQL statements;}... END
```

PSM : Persistent Stored Modules

- Module $\simeq$ Paquetage
- Contient
  - Eventuellement des tables locales temporaires
  - Des procédures et fonctions
- Langage de programmation :
  - Déclaration de variables
  - affectation (SET)
  - structures de contrôle (IF, CASE, LOOP, WHILE, REPEAT, FOR)
  - Traitement des erreurs (à la manière de SQL-intégré)
- On peut écrire un module dans un autre langage que SQL : Ada, C, ...

Existant

source Wikipédia

Source	Common Name	Full Name
ANSI/ISO Standard	SQL/PSM	SQL/Persistent Stored Modules
Interbase/Firebird	PSQL	Procedural SQL
IBM	SQL PL	SQL Procedural Language (implements SQL/PSM)
Microsoft/Sybase	T-SQL	Transact-SQL
MySQL	SQL/PSM	SQL/Persistent Stored Module (as in ISO SQL :2003)
Oracle	PL/SQL	Procedural Language/SQL (based on Ada)
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL Structured Query Language (based on Oracle PL/SQL)
PostgreSQL	PL/PSM	Procedural Language/Persistent Stored Modules (implements SQL/PSM)

Conclusion

- Les ajouts procéduraux (PSM + triggers) existent depuis longtemps dans les SGBD : il va falloir un peu de temps pour que la norme soit vraiment respectée.
- Les aspects objet-relationnels sont utilisés surtout pour des applications spécifiques (bases de données techniques, géographiques, ...)
- Pour la persistance d'objets issus de programmes, on utilise surtout le mapping objet-relationnel, i.e. on stocke des objets dans des bases relationelles (cf Hibernate).