Chapitre 2

Bases de données objet-relationnelles

Vers les SGBD orientés objet

- Besoin d'un SGBD capable de traiter
  - des éléments de structure complexe
  - des opérations (méthodes) sur les éléments
  - des pointeurs reliant les éléments (pour de l'héritage par exemple)
- Deux manières d'utiliser l'objet dans les SGBD
  - On part des langages objet dans lesquels on intègre les notions des SGBD (persistance des données, aspect multi-utilisateurs, ...). Ce sont les SGBD orientés objet : O2 (basé sur C++)
  - On part des SGBD relationnels dans lesquels on insère des notions objet. Ce sont les SGBD relationnels objet : ORACLE (SQL 3)
- Les SGBDOO sont plus "propres" du point de vue objet et les mieux adaptés pour traiter les objets mais ils sont complètement absents du monde professionnel
- Les SGBDRO sont basés sur des SGBD robustes et éprouvés répandus dans le monde professionnel mais qui ne sont pas prévus pour gérer l'objet

Caractéristiques du modèle relationnel objet

- Un modèle qui intègre des éléments de structure complexe
- Une relation sera en NF2 (Non First Normal Form) et pourra contenir un attribut composé d'une liste de valeurs ou de plusieurs attributs
- Exemple : table (assurance d'un véhicule)

N° contrat	Nom	Adresse	Conducteurs	Accidents
1111	J. CHIRAC	Elysée 75000 PARIS	Nom / Age Bernadette / 70 Claude / 75	Personne / Date Nicolas / 2005 Lionel / 2002
...	...	...	...	...

- Par rapport au modèle relationnel standard
  - on a moins besoin d'aplatir les données, ici on a une seule table avec des tables imbriquées au lieu de 3 tables
  - on a moins besoin de faire des jointures pour récupérer les informations sur les accidents du contrat 1111, on accède simplement à l'attribut accidents

Création de types

- Les types utilisés dans les BDOO sont
  - les types standards existant dans les BD classiques : VARCHAR, NUMBER...
  - les types utilisateurs
    - définis par le concepteur de la base
    - utilisés comme des types standards
- On appelle aussi ces types utilisateurs des types objet car ils ont une structure complexe et peuvent contenir des opérations (méthodes)
- Exemple de création de type objet
- Exemple d'objet de type t_adresse

```
CREATE TYPE t_adresse AS OBJECT
 ( num NUMBER,
   rue VARCHAR(30),
   ville VARCHAR(20),
   codepostal CHAR(5)
);
```

t_adresse (2, 'Bd Lavoisier', 'ANGERS', '49000')

NB : c'est un constructeur d'objet

- On peut ensuite utiliser ce type objet (ou type utilisateur)
  - soit pour définir une table relationnelle standard
  - soit pour définir une table relationnelle objet (ou table objet relationnelle)
  - soit pour définir d'autres types objet qui contiennent cette structure

Utilisation d'un type utilisateur

- Utilisation du type dans un autre type

```
CREATE TYPE t_personne AS OBJECT
(  nom VARCHAR(30),
   prenom VARCHAR(30),
   adresse t_adresse
 );
```

- Utilisation du type dans une table relationnelle (comme un type prédéfini)

```
CREATE TABLE employes
( num NUMBER,
  dept NUMBER,
  salaire NUMBER,
  adresse t_adresse, -- type objet
  nom VARCHAR(30),
  PRIMARY KEY num -- on peut définir des contraintes habituelles sur la table
);
```

Manipulation d'une table

- Insertion dans une table relationnelle
  - on utilise le constructeur d'objet t_adresse pour écrire l'adresse

```
INSERT INTO employes VALUES (1000, 15, 2000, t_adresse (2, 'Bd Lavoisier', 'ANGERS', '49000'), 'toto') ;
```

num	dept	salaire	adresse (num, rue, ville, cp)	nom
1000	15	2000	t_adresse (2, 'Bd Lavoisier', 'ANGERS', '49000')	toto

- Interrogation d'une table relationnelle

```
SELECT * FROM employes ;
```

```
SELECT e.adresse FROM employes e ; -- Préciser un alias de table
```

adresse (num, rue, ville, cp)
t_adresse (2, 'Bd Lavoisier', 'ANGERS', '49000')

```
SELECT e.adresse.num FROM employes e ;
```

adresse.num
2

Création d'une table objet relationnelle

- Une table objet relationnelle est une table qui contient des éléments de type objet
  - Chaque élément est identifié par un numéro appelé OID (Object Identifier)
- Exemple avec le type t_adresse `CREATE TABLE adresses OF t_adressse ;`
  - On a une relation dont chaque élément est un objet de type t_adresse
- On peut voir la relation de deux manières différentes
  - *vision objet*

t_adresse(…)
OID1 | num1 | rue1 | ville1 | cp1
OID2 | num2 | rue2 | ville2 | cp2
…

  - *vision relationnelle*

	num	rue	ville	cp
OID1	num1	rue1	ville1	cp1
OID2	num2	rue2	ville2	cp2
…	…	…	…	…

- On peut ecrire des contraintes comme en relationnel standard lors de la création de la table, on peut définir des clés, etc.

```
CREATE TABLE adresses OF t_adresse (ville DEFAULT 'ANGERS') ;
```

Insertion dans une table objet relationnelle

- On a deux manières d'insérer des n-uplets
  - soit avec le constructeur de type (vision objet)

```
INSERT INTO adresses VALUES ( t_adresse(30, 'Bd Foch', 'ANGERS', '49000') ) ;
```

    - on insère un objet avec constructeur de type et valeurs des champs du type objet
  - soit en précisant chacun des champs (vision relationnelle)

```
INSERT INTO adresses VALUES (30, 'Bd Foch', 'ANGERS', '49000') ;
```

    - on précise les valeurs des différents attributs de la table relationnelle

- Les deux requêtes sont équivalentes
  - on insère un n-uplet
  - un OID lui est attribué lors de l'insertion

Interrogation dans une table objet relationnelle

- On peut accéder aux valeurs comme dans le cas du relationnel standard

```
SELECT * FROM adresses ;
```

```
SELECT a.num, a.rue, a.ville, a.cp FROM adresses a ;
```

  - -- a est un alias de table
  - En accédant aux valeurs des attributs

num	rue	ville	cp
2	Bd Lavoisier	ANGERS	49000
…	…	…	…

- On peut accéder aux objets

```
SELECT VALUE(a) FROM adresses a ;
```

  - -- a est un alias de table et VALUE est un mot clé pour récupérer les objets
  - *En accédant aux objets*

VALUE (a) (num, rue, ville, cp)
t_adresse(2, 'Bd Lavoisier', 'ANGERS', 49000)
…

- On peut accéder aux OID

```
SELECT REF(a) FROM adresses a ;
```

  - -- a est un alias de table et REF est un mot clé pour récupérer les OID
  - *En accédant aux OID : On obtient une relation de référence où chaque référence correspond à un code assez long fait de chiffres et de lettres*

Utilisation des références (ou pointeurs)

- TABLE employes

num	dept	salaire	adresse				nom
1	15	2000	2	Bd Lavoisier	ANGERS	49000	toto
6	13	1000	2	Bd Lavoisier	ANGERS	49000	titi
9	12	3000	10	Bd Foch	ANGERS	49000	tata

- Plutôt que cette représentation qui pose des problèmes de redondance, de maintien de cohérence lors des mises à jours, on préfèrera la représentation suivante
  - TABLE employes (avec des OID d'objets de la table adresses)

num	dept	salaire	adresse	nom
1	15	2000	ABC1234	toto
6	13	1000	ABC1234	titi
9	12	3000	XYZ9999	tata

  - TABLE adresses — table objet relationnelle

	num	rue	ville	cp
ABC1234	2	Bd Lavoisier	ANGERS	49000
XYZ9999	10	Bd Foch	ANGERS	49000

Définition de types utilisant des références

❖ Exemple

- TABLE pays Relationnelle (avec des OID d'objets de la table villes)

nom	capitale *(référence)*	population
FRANCE	ABC123	60 000 000
ITALIE	XYZ999	57 000 000

- TABLE villes — objet relationnelle

	nom	population
ABC123	Paris	2 000 000
XYZ999	Rome	2 700 000

- En créant une table relationnelle

```
CREATE TYPE t_ville AS OBJECT (
nom VARCHAR(10),
population NUMBER ) ;
```

```
CREATE TABLE villes OF t_ville ;
```

```
CREATE TABLE pays (
nom VARCHAR(30),
capitale REF t_ville,
population NUMBER ) ;
```

Définition de types utilisant des références

- Ou en créant un autre type objet et une table objet relationnelle
  - TABLE pays objet relationnelle (avec des OID d'objets de la table villes)

	nom	capitale *(référence)*	population
AAA111	FRANCE	ABC123	60 000 000
AAA222	ITALIE	XYZ999	57 000 000

```
CREATE TYPE t_ville AS OBJECT (
nom VARCHAR(10),
population NUMBER ) ;
```

```
CREATE TYPE t_pays AS OBJECT (
nom VARCHAR(30),
capitale REF t_ville,
population NUMBER );
```

```
CREATE TABLE pays OF t_pays ;
```

- Un objet tout seul n'a pas d'OID, c'est un objet d'une table objet relationnelle qui possède un OID

Insertion de références dans les tables objet relationnelles

❖ Table pays

	nom	capitale *(référence)*	population
AAA111	FRANCE	ABC123	60 000 000
AAA222	ITALIE	XYZ999	57 000 000

❖ Table villes

	nom	population
ABC123	Paris	2 000 000
XYZ999	Rome	2 700 000

```
INSERT INTO villes VALUES ( 'Paris', 2000000 ) ;
INSERT INTO villes VALUES ( 'Rome', 2700000 ) ;
INSERT INTO pays (nom, population) VALUES ( 'FRANCE',60000000 ) ;
```

```
UPDATE pays SET capitale = ( SELECT REF(v) FROM villes v WHERE v.nom =
'Paris' ) WHERE nom = 'FRANCE' ;
```

```
INSERT INTO pays
SELECT 'ITALIE', REF(v), 57000000
FROM villes v
WHERE v.nom = 'Rome' ;
```

Interrogation de tables utilisant des références

- Les données sont interrogées comme si elles étaient physiquement dans la table

- Exemples

```
SELECT * FROM pays ;
```

```
SELECT p.nom, p.capitale.nom, p.population
FROM pays p ;
```

-- le résultat est la table pays ci-dessous

nom	capitale.nom	population
FRANCE	Paris	60 000 000
ITALIE	Rome	57 000 000

- Autre fonction sur les objets
  - DEREF qui renvoie un objet à partir de sa référence

```
SELECT DEREF ( p.capitale ) FROM pays p ;
```

-- le résultat est t_ville('Paris', 2 000 000), t_ville('Rome', 2 700 000)

```
SELECT DEREF ( REF(v) ) FROM villes v ;
```

-- on obtient les objets de la table villes. C'est équivalent à la requête suivante

```
SELECT VALUE ( v ) FROM villes v ;
```

Les collections imbriquées

- Attribut sous forme d'une table
- Exemple

nom	prénom	liste des diplômes
…	…	
…	…	

- Il existe deux types de collections imbriquées
  - les tables imbriquées (NESTED TABLE) dont on ne fixe pas la taille à priori
  - les tableaux fixes (VARRAY)

Déclarations de types et de tables utilisant des tables imbriquées

- NB : Sous Oracle, il ne peut y avoir qu'un seul niveau d'imbrication

num_dep	budget	employes
1	100 000	
13	50 000	

-- type des éléments de la table imbriquée t_employe

```
CREATE TYPE t_employe AS OBJECT ( num_insee VARCHAR(20), nom VARCHAR(30),
age NUMBER );
```

-- type pour une table imbriquée t_employes

-- t_employe est le type des éléments de la table

```
CREATE TYPE t_employes AS TABLE OF t_employe ;
```

-- table relationnelle standard

-- table_emp : nom physique de la table imbriquée (ne sert jamais dans les requêtes)

```
CREATE TABLE departements
( num_dep NUMBER,
  budget NUMBER,
  employes t_employes
) NESTED TABLE employes STORE AS table_emp ;
```

Création des tables objet relationnelles utilisant des tables imbriquées

-- table objet relationnelle

```
CREATE TYPE t_departement AS OBJECT
( num_dep NUMBER,
  budget NUMBER,
  employes t_employes );
```

```
CREATE TABLE departements OF t_departement
NESTED TABLE employes STORE AS table_emp ;
```

-- STORE AS table_name à préciser lors de la création de la table

	num_dep	budget	employes
OID1	1	100 000	
OID2	13	50 000	

Insertion dans une relation utilisant une table imbriquée

- Pour insérer un élément qui comporte une table imbriquée, on peut, comme pour les autres objets, utiliser le constructeur de type

```
INSERT INTO departements VALUES (1, 200 000,
t_employes(t_employe(12345, 'toto', 25), t_employe(2222, 'titi', 28))) ;
```

-- t_employes est le constructeur de type de la table employe

-- t_employe est le constructeur de type des éléments de la table imbriquée

-- (12345, 'toto', 25) et (2222, 'titi', 28) sont les valeurs des attributs de l'élément

-- insertion d'une table imbriquée vide

```
INSERT INTO departements VALUES (2, 100 000, t_employes()) ;
```

-- insertion sans création de la table imbriquée

```
INSERT INTO departements (numdep, budget) VALUES (4, 100 000) ;
```

	num_dep	budget	employes
OID	1	200 000	
OID	2	100 000	
OID	4	100 000	null

Insertion dans une table imbriquée

❖ Commande particulière pour insérer dans une table imbriquée : THE

```
INSERT INTO THE ( SELECT employes FROM departements WHERE numdep = 1 )
VALUES ( t_employe (789, 'tutu', 20) ) ;
```

-- (SELECT …) est la table imbriquée dans laquelle on insère l'élément
-- employes est l'attribut de la table qui contient la table imbriquée
-- numdep = 1 est la condition du n-uplet dont on veut modifier la table
-- t_employe (789, 'tutu', 20) est l'élément à insérer dans la table imbriquée

Attention : on ne peut insérer que dans une seule table imbriquée

Insertion dans une table imbriquée

	num_dep	budget	employes
OID	1	200 000	
OID	2	100 000	
OID	4	100 000	null

❖ Que se passe-t-il avec les requêtes suivantes ?

```
INSERT INTO THE ( SELECT employes FROM departements
WHERE numdep = 1 ) VALUES ( t_employe(5432, 'XXX', 27) ) ;
```

-- on insère l'élément

```
INSERT INTO THE ( SELECT employes FROM departements
WHERE numdep = 4 ) VALUES ( t_employe(987, 'YYY', 50) ) ;
```

-- ça ne marche pas car la table imbriquée n'exite pas

Insertion dans une table imbriquée

- Pour que ça marche, Il faut créer la table imbriquée au préalable
  - On met à jour le n-uplet pour rendre l'insertion possible

```
UPDATE departements SET employes = t_employes() WHERE numdep = 4 ;
```

  - puis on exécute la requête d'insertion

```
INSERT INTO THE ( SELECT employes FROM departements
WHERE numdep = 4 ) VALUES ( t_employe(987, 'YYY', 50) ) ;
```

  - Ou on met à jour en insérant l'élément

```
UPDATE departements SET employes = t_employes(t_employe(987,
'YYY', 50) WHERE numdep = 4 ;
```

Interrogation des tables utilisant des tables imbriquées

❖ Pour la table objet-relationnelle

```
SELECT * FROM departements ;
```

--On obtient les valeurs des attributs comme habituellement avec les tables imbriquées sous forme de constructeurs de type

❖ Pour les tables imbriquées, on utilise THE comme pour l'insertion

```
SELECT e.nom FROM THE ( SELECT employes FROM departements
WHERE numdep = 1 ) e ;
```

-- on doit utiliser un alias de table (e)
-- e.nom est un attribut du type t_employe
-- (SELECT ...) est une table imbriquée

❖ Remarque : Ici aussi, il faut sélectionner une seule table imbriquée

Récupérer les informations de plusieurs tables imbriquées simultanément

❖ On utilise la commande CURSOR, par exemple :

```
SELECT d.numdep, CURSOR ( SELECT e.nom FROM TABLE (employes) e )
FROM departements d ;
```

--On obtient la liste des départements avec, pour chaque département, la liste des noms des employés

numdep	employes.nom
1	nom11
	nom12
	nom13
2	nom21
	nom22
....	...

Utilisation de plusieurs tables imbriquées

- Comment faire si on veut des pays qui contient une liste de régions, chaque région contenant une liste de villes ?

nom	population	régions
FRANCE	60 000 000	Pays de la Loire: Nantes, Angers … / Ile de France: Paris, Versailles, Créteil …

- On va utiliser plusieurs tables relationnelles objet et les relier par des références, On utilise donc des imbriqués de références à des objets

nom	population	régions
FRANCE	60 000 000	ABC123, XYZ987

	nom	…	Villes
ABC123	Pays de la Loire	…	Nantes, Angers …
XYZ987	Ile de France	…	Paris, Versailles, Créteil …

Les tableaux fixes (VARRAY)

- C'est une collection limitée, ordonnée d'éléments de même type
- Un tableau fixe permet
  - d'avoir plusieurs niveaux d'imbrication (contrairement aux NESTED TABLE qui nécessitent l'utilisation de références)
  - d'accéder aux éléments par leur numéro
- Mais on ne peut pas accéder à un élément particulier du VARRAY dans une requête SQL standard, il faut utiliser un bloc PL/SQL (langage procédural qui intègre les requêtes SQL) contrairement aux NESTED TABLE qui se comportent comme des tables relationnelles standard
- Exemple

```
CREATE TYPE tvadresses AS VARRAY(2) OF tadresse ;
```

-- tadresse est le type des éléments du VARRAY, type utilisateur défini
-- 2 est la taille du VARRAY
-- tvadresses est le type VARRAY défini

Utiliser un seul élément

- Si on veut travailler sur un seul élément, il faut utiliser un bloc PL/SQL

```
CREATE TYPE tadresse AS OBJECT(
Rue VARCHAR(30),
ville VARCHAR(20)
);
```

```
CREATE TYPE tvadresses AS VARRAY(2) OF tadresse ;
```

```
-- ici, une table relationnelle
CREATE TABLE etudiants (
code VARCHAR(10),
nom VARCHAR(30),
adresses tvadresses
) ;
```

- Ajouter l'adresse ('Bd Foch', 'Paris') à l'étudiant nommé 'Dom'

```
-- déclaration des variables
DECLARE
lesadr tvadresses;
-- instructions
BEGIN
-- initialisation de la variable lesadr
SELECT adresses INTO lesadr FROM etudiants WHERE nom = 'Dom' ;
-- modification de la deuxième valeur
lesadr(2) := tadresse('Bd Foch', 'Paris') ;
-- mise à jour de la relation
UPDATE etudiants SET adresses = lesadr WHERE nom = 'Dom' ;
END ;
```

Les méthodes

- Une méthode (ou opération) est la modélisation d'une action applicable sur un objet
- On déclare les méthodes
  - soit au début, lors de la déclaration de l'objet
  - soit plus tard, avec commande ALTER TYPE
- Le corps de la méthode correspond aux opérations effectuées sur l'objet, il peut faire référence à l'objet concerné grâce à SELF

Utilisation des méthodes

❖ Exemple : type objet contenant une fonction

```
CREATE TYPE tpersonne AS OBJECT (
nom VARCHAR(10), datenaiss date,
MEMBER FUNCTION age RETURN number
-- le type contient une fonction de nom age sans paramètre qui retourne un nombre
)
```

-- *implémentation des méthodes*

```
CREATE TYPE BODY tpersonne AS MEMBER FUNCTION age
RETURN number
IS n number ;
BEGIN
n := TRUNC((SYSDATE - SELF.datenaiss)) ; /* TRUNC pour la partie entière*/
RETURN n ;
END age ;
END ;
```

-- Avec personnes table objet-relationnelle de type tpersonne

```
CREATE TABLE personnes OF tpersonne
SELECT p.age() FROM personnes p ;
```

Modèle OR : notations – types

- **Table relationnelle**

  <nom_rel>(<nom_attr1> <nom_type1>, ... <nom_attrj> <nom_typej>)

Exemple : Departements (nomdpt varchar(10), resp varchar(10), projet type_projet)

- **Type structuré**

  <nom_type>(<nom_attr1> <nom_type1>, ... <nom_attri> <nom_typei>)

Exemple : type_projet (refp varchar(5), deb date, durée number)

- **Table objet_relationnelle**

  <nom_rel>$^{<nom\_type>}_{O,}$ clé : {attr_clé}

Exemple : Departements$^{type_dpt}_{O}$, clé : {nomdpt}

- **Pointeur**

  <nom_attr> @<nom_type>

Exemple : Departement (nomdpt varchar(10), resp @type_resp, projet type_projet)

- **Collection**

  <nom_attr> {<nom_type>}

Exemple : Departement (nomdpt varchar(10), resp varchar(10), projets {type_projet})