Persistance des Objets

Hala Skaf-Molli
Maître de Conférences UHP
www.loria.fr/~skaf

Janvier 2008

Plan

- Introduction
- Bases de données Objet relationnelles
- L'API de base pour la persistance en Java (JDBC)
- Correspondance Objet-relationnel
- Patterns pour la persistance (DAO)
- Outils et frameworks de mapping : Hibernate

Persistance des objets

- A la fin d'une session d'utilisation d'une application orientée objet toutes les données des objets existant dans la mémoire vive de l'ordinateur sont perdues
- Rendre persistant un objet c'est sauvegarder ses données sur un support non volatile de telle sorte qu'un objet identique à cet objet pourra être recréé lors d'une session ultérieure

Persistance des objets

- Une application écrite avec un langage objet (JAVA) qui utilise une base de données pour la persistance des données des objets
  - Bases de données orienté objets (SGBDOO)
  - Bases de données objet-relationnelles (SGBDOR)
  - Bases de données relationnelles (SGBDR)
    - Persistance à la main par JDBC
    - Quels sont les moyens disponibles pour effectuer cette persistance? OR mapping

Pourquoi les BD relationnelles ?

- Position dominante
- Une théorie solide et des normes reconnues
- Grande facilité et efficacité pour effectuer des recherches complexes dans des grandes bases de données
- Fonctionnalités de SGBD
  - Cohérence de données: contraintes d'intégrité
  - Gérer les accès concurrents, transactions
  - Partage de données

Pourquoi pas un SGBD Objet ?

- Principale raison ‘legacy’: de très nombreuses bases relationnelles en fonctionnement
- Manque de normalisation pour les SGBDOO ; trop de solutions propriétaires
- Peu d'informaticiens formés aux SGBDOO

L’objet et le relationnel sont 2 paradigmes bien différents

- Faire la correspondance entre les données modélisées par un modèle objet et par un modèle relationnel n’est pas simple
- Le modèle relationnel est moins riche que le modèle objet
- Par exemple, pas d’héritage, ni de références, ni de collections (pas d’attributs multi-valués) dans le modèle relationnel

Bases de données objet-relationnelles

Hala Skaf-Molli
Maître de Conférences UHP
www.loria.fr/~skaf

Modèle Objet-relationnel

- Le modèle Objet-Relationnel (OR) reprend le modèle relationnel en ajoutant quelques notions qui comblent les plus grosses lacunes du modèle relationnel.
- Le modèle de données :
  - Relations et on ajoute des possibilités de définir de types complexes avec de méthodes.
- Fusionner le modèle relationnel avec le modèle objets

Pourquoi étendre le modèle relationnel ?

1. L'impossibilité de créer de nouveaux types implique un manque de souplesse et une interface difficile avec les applications orientées objet
2. L'OR permet de définir de nouveaux types utilisateur User-defined-Type UDT simples ou complexes. UDT avec des fonctions ou procédures associées comme dans les classes des langages objet
3. L'OR supporte l'héritage de type pour profiter du polymorphisme et faciliter la réutilisation

En bref, ajouter des possibilités objets au modèle relationnel

Évolution de SGBD

- Échec de bases de données orienté objet
  - Efficacité, héritage
- L'extension OR des bases de données relationnelles permet de capter toutes les avantages de OO tout en gardant la relation..

SQL-99 et Oracle

- La norme SQL-99 (SQL3) reprend beaucoup d'idées du modèle OR.
- La norme est récente, différentes implémentations dans les SGBD.
- Dans ce cours, on prend comme exemple l'extension objets dans Oracle.

Les nouvelles possibilités de l'OR

- **Nouveaux types** complexes avec des fonctions pour les manipuler
  - la colonne d'une table peut contenir une collection (ensemble, sac, liste).
  - la ligne d'une table peut être considérée comme un objet, avec un identificateur (Object Identifier OID).
  - L'utilisation de références aux objets permet d'accéder rapidement aux objets référencés.
- L'extension du langage SQL (SQL3 ou SQL99) pour la recherche et la modification des données.

Définition de types

- Type défini par l'utilisateur (User Defined Type UDT) : c'est une classe avec une structure et des méthodes. UDT peut-être le type :
  1. d'un attribut dans la déclaration un autre UDT,
  2. d'une table i.e. le type de ses tuples,
  3. d'un attribut (colonne) dans une table.
- Type collection…

Types distincts

- Ces types permettent de différencier les domaines de colonnes. Ils sont formés à partir de types de base.
- Example:

```
create type poids as integer ;
create type age as integer;

create table Personne (
personneID varchar(20) primary key,
personneAge age,
personnePoids poids);
```

- Ces types s'utilisent exactement avec les mêmes instructions que le type de base sous-jacent.
- Je n'ai pu tester dans Oracle 10g.

Création d'un type utilisateur

- Create type T as <attributes and methode declarations>
- Exemple:

```
CREATE TYPE Customer_objtyp AS
 (CustNo NUMBER,
 CustlName VARCHAR2(200),
Custfname VARCHAR2(200)
);
```

UDT dans Oracle

CREATE OR REPLACE TYPE <type_name>
AS OBJECT (
<attribute> <attribute data_type>,
<subprogram spec>)
<FINAL | NOT FINAL> <INSTANTIABLE | NOT INSTANTIABLE>;

Remarques :

1. Dans SQL-99 pas OBJECT après le AS.
2. Un type ne contient pas de contrainte d'intégrité
3. Je peux utiliser la commande "create or replace"

Exemple

```
CREATE TYPE Customer_objtyp AS
   OBJECT
 (CustNo NUMBER,
 CustlName VARCHAR2(200),
Custfname VARCHAR2(200)
);
```

Types structurés

Chaque UDT a :

2. un constructeur de même nom que le type,
3. attributs (variables d'instances),
4. fonctions et procédures (méthodes). Elles peuvent être écrites en SQL ou en un autre langage

Les méthodes dans l'UDT

- Les UDT sont plus qu'une structure de données, ils ont des méthodes.
- Les méthodes sont déclarées dans **create type.**
- Elles sont définies dans **create type body** :
  - on utilise la syntaxe de PL/SQL.
  - la variable self fait référence à l'objet auquel j'applique la méthode.

Déclaration

- On déclare les méthodes dans la définition de UDT

```
CREATE TYPE Person_objtyp AS OBJECT
    (PersNo NUMBER,
    PerslName VARCHAR2(200),
   Persfname VARCHAR2(200),
   MEMBER FUNCTION getPersNo RETURN NUMBER );
   /
```

Définition

- La définition de méthode (le style d'Oracle)

  CREATE TYPE BODY <type name> AS

  <method definitions = PL/SQL

  procedure definitions, using

  "MEMBER FUNCTION" in place of

  "PROCEDURE">

  END;

- /

Exemple

```
create type body Person_objtyp as
member function getPersNo return umber
   is
begin
return PersNO;
end;
end;
```

Utilisation de méthodes

```
select a.salaire, a.salaireInEuros(1.44)
from Acteurs a
where a.name='Jackson' ;
```

- Dans l'exemple, ci-dessus la méthode salaireInEuros est définie dans la classe Acteurs.

Héritage (création de sous type)

- Les types supportent l'héritage multiple avec le mot-cl´e UNDER :

CREATE OR REPLACE TYPE <type_name>

UNDER <supertype_name> (

<attribute> <data_type>,

<inheritance clause> <subprogram spec>, <pragma clause>)

<FINAL | NOT FINAL> <INSTANTIABLE | NOT INSTANTIABLE>;

/

Exemple

Super type:

```
create type Personne_objtyp as Object (
nom varchar(20),
rue varchar(30),
ville varchar(20),
Member Function getNom return varchar)
NOT FINAL;
```

- Sous type:

```
create type Etudiant_Objtype under Personne_Objtype (
note float);
```

- Le type est final par défaut. Le super type doit être NOT

FINAL.

- On peut rendre un type NOT FINAL, en utilisant :

alter type tye_name NOT FINAL;

Types NOT INSTANTIABLE

CREATE TYPE Personne_objtyp AS OBJECT(...)
NOT INSTANTIABLE NOT FINAL;

CREATE TYPE Etudiant_Objtype UNDER
Personne_objtyp …

CREATE TYPE Employe_Objtype UNDER
Personne_objtyp …

Méthodes NOT INSTANTIABLE

```
CREATE TYPE Person_objtyp AS OBJECT (
 idno NUMBER,
name VARCHAR2(30),
phone VARCHAR2(20),
NOT INSTANTIABLE MEMBER FUNCTION get_idno
    RETURN NUMBER)
NOT INSTANTIABLE NOT FINAL;
/

ALTER TYPE Person_objtyp INSTANTIABLE;
```

Ajout d'attribut et de méthode

- Ajout d'un attribut à un type

  alter type Person_objtyp

  add attribute date_naissance date cascade ;

  Cascade permet de propager aux tables déjà construites à partir du type.

- Ajout d'une méthode/fonction à un type

  alter type Person_objtyp

  add member function age return integer cascade;

Vues du dictionnaire des données

- USER_TYPES pour les types
- USER_coll_TYPES, pour les collections
- USER_OBJECT_TABLES pour les tables objet relationnelles
- USER_VARRAYS : type collection

Exemple

SQL> desc user_types

1 TYPE_NAME NOT NULL VARCHAR2(30)
2 TYPE_OID NOT NULL RAW(16)
3 TYPECODE VARCHAR2(30)
4 ATTRIBUTES NUMBER
5 METHODS NUMBER
6 PREDEFINED VARCHAR2(3)
7 INCOMPLETE VARCHAR2(3)
8 FINAL VARCHAR2(3)
9 INSTANTIABLE VARCHAR2(3)
10 SUPERTYPE_OWNER VARCHAR2(30)
11 SUPERTYPE_NAME VARCHAR2(30)
12 LOCAL_ATTRIBUTES NUMBER
13 LOCAL_METHODS NUMBER
14 TYPEID

Tables OR

- Les données d'un type sont persistantes seulement si elles sont rangées dans une table.
- Il est possible de créer une table à partir d'un type.
- Syntaxe :
  - create table <table name> of <type name> ;

Création d'une table OR

create table Person_objtable of Person_objtype;

On peut ajouter des contraintes d'intégrité :

```
create table Person_objtable of Person_objtype
(constraint pkctr primary key(nom));

create table Person_objtable of Person_objtype
(constraint pkctr primary key(nom))
OBJECT IDENTIFIER IS PRIMARY KEY;
```

- On ajoute les contraintes dans la table et pas dans le type, pourquoi ?

Héritage de tables

- Une table peut hériter d'une ou plusieurs tables. Pas supporté par Oracle 10g.

Caractéristiques d'une table Objet-Relationnel

- Une table est une table OR si elle a été construite à partir d'un type (create table .. OF).
- Les lignes de ces tables sont considérées comme des objets avec un identifiant (OID).
- On peut utiliser des références pour désigner les lignes de ces tables (pas possible pour les autres tables)
- Vues du dictionnaire
  - USER_OBJECT_TABLES pour les tables objet relationnelles

Insertion des données

- Insertion simple :
  - insert into Person_objtable values(1,'Dupond', 0383000,'22-dec-02');
- Insertion avec constructeur
  - On peut aussi utiliser le "constructeur du type" avec lequel la table a été

  insert into Person_objtable values(Person_objtype((1,'Dupond', 0383000,'22-dec-02'));

Modifications

- On peut utiliser la notation pointée comme en SQL92 :

  update Person_objtable

  set Person_objtable.phone= 0666666

  where Person_objtable .idno= 1;

- SQL-99 fournit aussi la notation " .. " pour désigner un attribut d'une colonne de type structuré

  update Person_objtable

  set Person_objtable.adresse..numero = 12

  where Person_objtable.idno = 'titi';

Appel d'une procédure ou d'une fonction

```
select e.nom, age(e)    // Le " this " est passé
   en paramètre
from Person_objtable e
where age(e) < 40
```

- Sous Oracle :

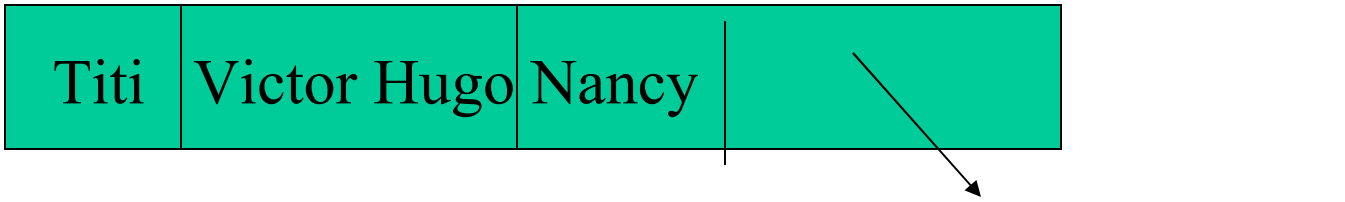
```
select e.name, e.age()
from Person_objtable e
where e.age() < 40
```

Références

- On peut indiquer dans la définition d'un type qu'un attribut contient des références (et non des valeurs) à des données d'un autre type ;
- la syntaxe est "REF nom-du-type" : (C'est un pointeur vers un objet de type nom-du-type)

Exemple

```
create type Employe_objtype as Object (
nom varchar(20),
rue varchar(30),
ville varchar(20),
dept REF dep_objtype);
```



Références

- Si T est un type alors REF T est le type d'une référence à T. C'est un pointeur vers l'identité d'objet (OID) de type T.
- Dans les langages OO l'OID n'est pas visible à l'utilisateur, tandis que REF est visible, mais c'est de ''charabia''.

Exemple de select avec référence

- La notation pointée permet de récupérer les attributs d'un type dont on a un pointeur
- La suivie de REF est implicite dans Oracle
- Exemple:
  - Lieu de travail des employés (avec Oracle).
  - create table Employe_objtable of Employe_objtype;

```
select e.nom, e.dept.lieu
from Employe_objtable e
```

Exemples

- Pas correcte

  select nom, dept.lieu

  from Employe_objtable

Pas correcte

  select Employe_objtable.nom

  from Employe_objtable

  C'est une bonne habitude d'utiliser un alias quand on utilise les fonctionnalités OR.

Le même exemple en SQL-99 :

select e.nom(), e.dept.lieu()

from employe e

- Attention, l'alias e est indispensable
- Dans SQL-99, chaque attribut d'un UDT a deux méthodes :
- Un générateur (get) et un mutator (set). Ces méthodes ont les mêmes noms que les attributs.
- Le générateur n'a pas d'arguments. Le mutator prend en argument la nouvelle valeur

Insertions avec référence

- La référence est un pointeur vers NULL
  - insert into employe values (1230, 'Durand', NULL);
- Référence vers le dept numéro 10

```
insert into Employe_objtable (nom, dept)
select  'Dupond', REF(d)
from dept d
where dept.numDept = 10;
```

L'opérateur DEREF d'Oracle-Motivation

- Trouver le département de l'employé 'Dupond'?

```
select e.dept
from Employe_objtable e
where e.nom='Dupond';
```

- C'est légal mais e.dept c'est une référence donc c'est de 'Charabia.

Utilisation DEREF

- L'opérateur DEREF permet de voir la valeur de l'objet de type departement_objtype.

```
select DEREF(e.dept)
from Employe_objtable e
where e.nom='Dupond';
```

Elle produit:

departementType(')..

Collections

- Oracle 10g n'offre que 2 types de collections
  - table imbriquée (NESTED TABLE)
    - une collection non ordonnée et non limitée
    - en nombre d'éléments
  - tableau prédimensionné (VARRAY)
    - une collection d'éléments de même type,
    - ordonnée et limitée en taille

Varray

```
CREATE TYPE phone_typ AS OBJECT (
    country_code VARCHAR2(2),
    area_code VARCHAR2(3),
    ph_number VARCHAR2(7)
    );
/
CREATE TYPE phone_varray_typ AS VARRAY(5) OF phone_typ;
/
CREATE TABLE dept_phone_list (
    dept_no NUMBER(5),
    phone_list phone_varray_typ);

 INSERT INTO dept_phone_list VALUES ( 100,
    phone_varray_typ( phone_typ ('01', '650', '5061111'),  phone_typ ('01',
    '650', '5062222'),  phone_typ ('01', '650', '5062525')));
```

Tables imbriquées

- Les valeurs d'un tuple peuvent être de relations
- Si T est un UDT, on peut créer un type S dont les valeurs sont de relations

  - Create type S as table of T;

Tables imbriquées

```
CREATE TYPE Employe_objtype as Object (
      nom varchar(20),
      rue varchar(30),
      ville varchar(20));

CREATE TYPE EmployeTableType AS TABLE OF
    Employe_objtype;

Create table Departement (
 nom varchar(30),
Addr vachar(50)
Empoles employeTableType);
```

Stockage des tables imbriquées

- Oracle ne stocke pas chaque table imbriquée séparément
- Il y a une seule relation R dans laquelle tous les tuples de toutes les tables imbriquées d'un attribut *N* sont stockées

Exemple

```
Create table Departement (
 nom varchar(30),
addr vachar(50),
employes employeTableType)
NESTED TABLE Empolyes STORE AS Emp_tab;
```

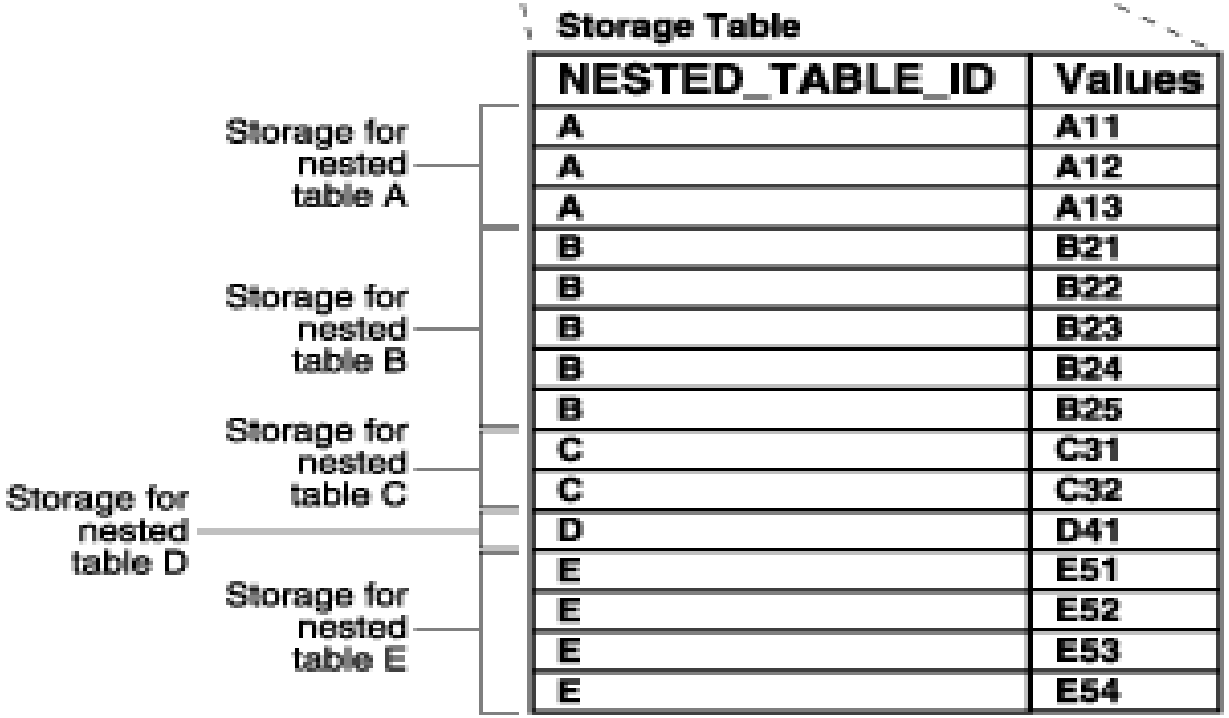
DATA1	DATA2	DATA3	DATA4	NT_DATA
...	...	...	...	A
...	...	...	...	B
...	...	...	...	C
...	...	...	...	D
...	...	...	...	E

Storage Table

NESTED_TABLE_ID	Values
B	B21
B	B22
C	C33
A	A11
E	E51
B	B25
E	E52
A	A12
E	E54
B	B23
C	C32
A	A13
D	D41
B	B24
E	E53

Nested tables storage
Source: doc Oracle

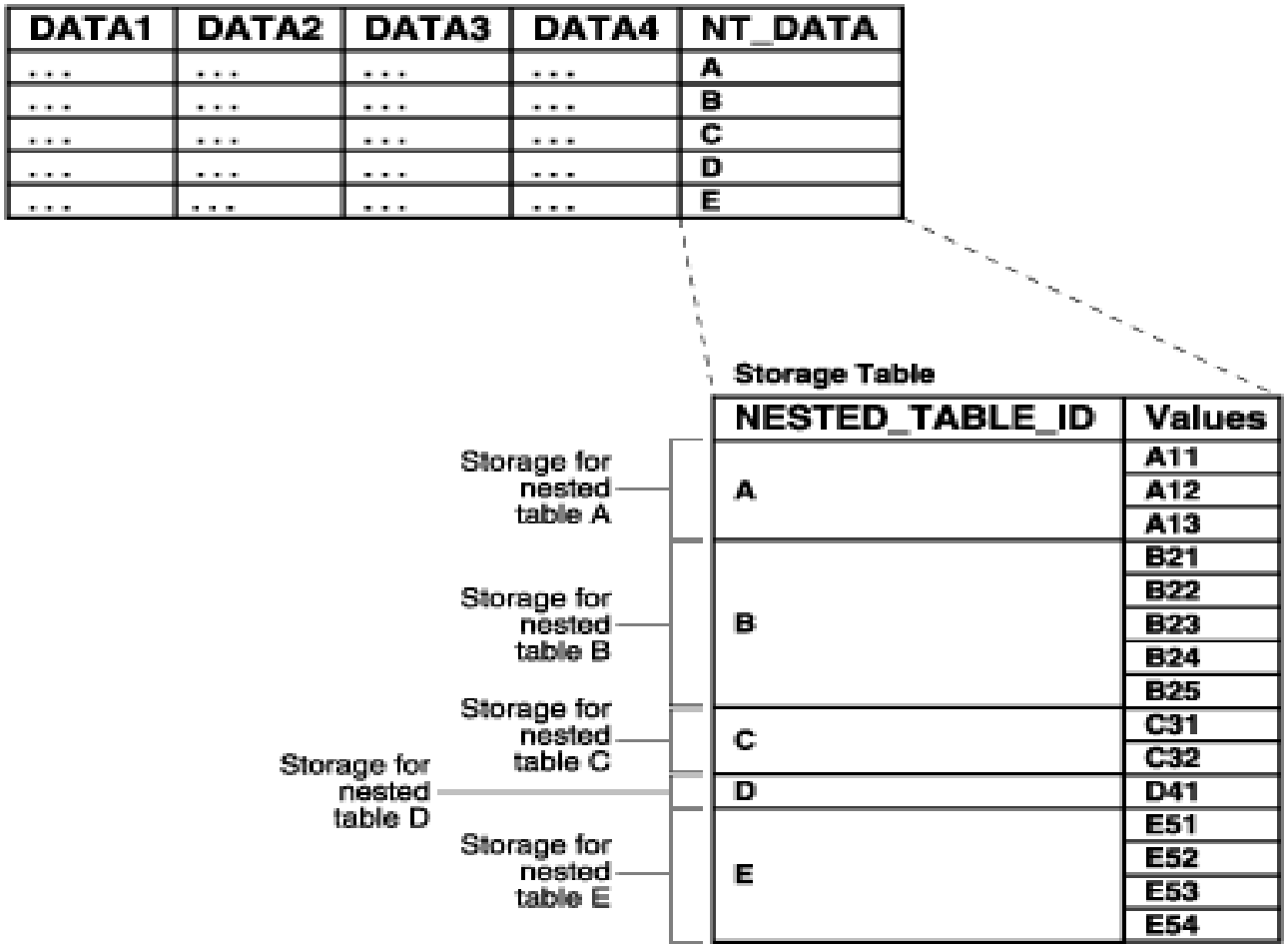
DATA1	DATA2	DATA3	DATA4	NT_DATA
. . .	. . .	. . .	. . .	A
. . .	. . .	. . .	. . .	B
. . .	. . .	. . .	. . .	C
. . .	. . .	. . .	. . .	D
. . .	. . .	. . .	. . .	E



Nested Table in index-organized table IOT Storage

Source Oracle

Nested Table in IOT Storage with Compression



Requêtes sur les tables imbriquées

- On peut manipuler les valeurs d'une table imbriquée comme n'importe quelle valeur.
- Les valeurs ont deux constructeurs:
  - Un pour le type de tuples de la table
  - Un pour la table

Insertion dans une table imbriquée

```
CREATE TYPE people_ ntabtyp AS TABLE OF
    person_objtyp;
/
CREATE TABLE people_tab (
  group_no NUMBER,
  people_column people_ ntabtyp )
 NESTED TABLE people_column STORE AS
    people_column_nt;


 INSERT INTO people_tab VALUES ( 100, people_
    ntabtyp ( person_objtyp(1, 'John Smith',
    '1-800-555-1212'), person_objtyp(2, 'Diane Smith',
    NULL)));
```

Exemple: Select

```
Select people_column
From people_tab
 where group_no = 1;

people_ ntabtyp( person_objtyp(1, 'John Smith',
   '1-800-555-1212'), person_objtyp(2, 'Diane
   Smith', NULL)));
```

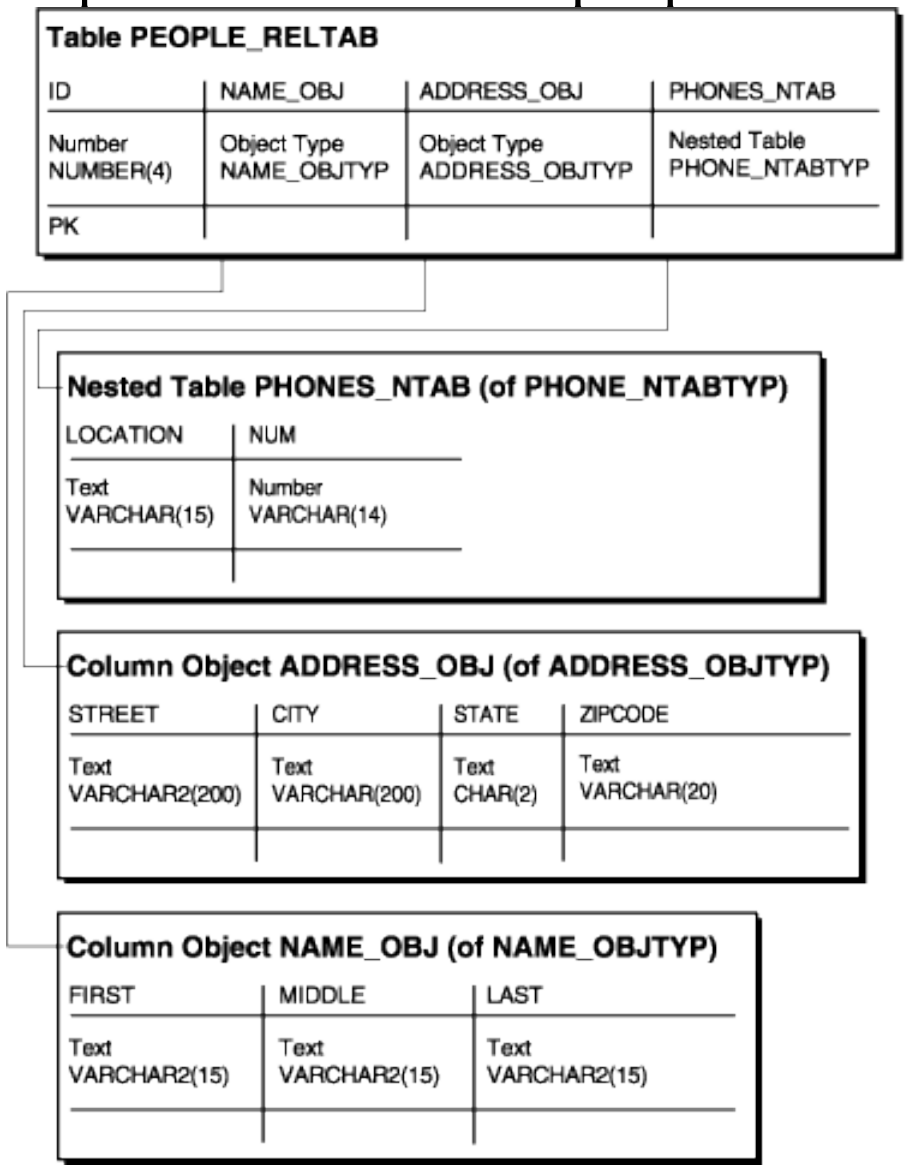
Exemple complet

```
CREATE TYPE name_objtyp AS OBJECT (
        first VARCHAR2(15),
        middle VARCHAR2(15),
       last VARCHAR2(15));
             /
CREATE TYPE address_objtyp AS OBJECT (
 street VARCHAR2(200),
 city VARCHAR2(200),
 state CHAR(2),
 zipcode VARCHAR2(20));
 /
CREATE TYPE phone_objtyp AS OBJECT (
location VARCHAR2(15),
num VARCHAR2(14));
/
CREATE TYPE phone_ntabtyp AS TABLE OF phone_objtyp;
```

Exemple complet

```
CREATE TABLE people_reltab (
   id NUMBER(4) CONSTRAINT
    pk_people_reltab PRIMARY KEY,
     name_obj name_objtyp,
 address_obj address_objtyp,
 phones_ntab phone_ntabtyp)
NESTED TABLE phones_ntab STORE AS
    phone_store_ntab;
```

Representation of the people_reltab Relational Table



Remarques

- Le modèle Objet-Relationnel assure la compatibilité ascendante : les anciennes applications relationnelles fonctionnent dans le monde OR
- Le produit open source d'Oracle TopLink permet de faire la correspondance entre objet et Objet-Relationnel.